DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de um Sistema de Monitoração e Validação para áreas de Transferência e Estocagem de uma Refinaria de Petróleo

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Bruno Martins Rahal

Florianópolis, Agosto de 2012

Desenvolvimento de um Sistema de Monitoração e Validação para áreas de Transferência e Estocagem de uma Refinaria de Petróleo

Bruno Martins Rahal

Este relatório foi julgado no contexto da disciplina
DAS 5511: Projeto de Fim de Curso
e aprovado na sua forma final pelo
Curso de Engenharia de Controle e Automação

Banca Examinadora:

Eng. Diogo Ferreira Pacheco, Me.
Orientador Radix

Professor Rômulo Silva de Oliveira, Dr.
Orientador acadêmico

Professor Ricardo José Rabelo, Dr.
Responsável pela disciplina

Professor Max Hering de Queiroz, Dr.
Avaliador

Fernando Corteccioni Nuñez Del Prado
Debatedor

Tárik Medeiros Siqueira
Debatedor

Agradecimento

Aos meus pais,

como se dissesse água.

Resumo

Um sistema de automação é um termo genérico e amplo, podendo englobar sistemas simples que realizam operações antes operadas por humanos com força bruta (como abrir e fechar um portão) até gigantecos sistemas fabris. Estes podem, inclusive, ser monitorados remotamente. Nestas operações, diversos tipos e uma enorme quantidade de dados são gerados. Mantê-los e saber utilizá-los com destreza é fundamental para otimizar uma operação.

Neste sentido, este trabalho buscou desenvolver um sistema de monitoração de dados de uma refinaria de petróleo, validando-os, para torná-los mais confiáveis e consistentes ao processo. Além disso, um centralizador de dados foi desenvolvido para o projeto, que busca evitar que sistemas se comuniquem com diversos outros, tornando um emaranhado de comunicação. Assim, os sistemas recorrem a esta fonte única de dados, quando necessitam de informação.

As atividades neste projeto fazem parte do escopo da disciplina DAS5511 – Projeto de Fim de Curso, do curso de graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina. Além disso, foi desenvolvido na empresa Radix – Engeharia e Software.

Inicialmente, são apresentadas a empresa, um motivador do projeto, além de objetivos e metodologia de trabalho. Após isso, conceitos básicos são apresentados, com toda a fundamentação base do trabalho. Por fim, é apresentada a solução, detalhando alguns aspectos do sistema, baseada na teoria previamente apresentada.

Palavras-chave: monitoração, validação, automação.

Abstract

An automation system is a generic and wide term, which can refer itself to simple systems that just do brute force operations, which was made by humans (like opening and closing a gate), but can also refer to huge manufacturing systems, that can even be remotely operated. Several kinds and an enormous quantity of data are generated in processes. Maintaining and knowing how to use them smartly is essential to optimize an operation.

In this sense, this paper describes the development of a monitoring system to the data of an oil refinery. This data is also validated, being more reliable and consistent with the process. Moreover, a data centralizer was developed, which aims to avoid that systems communicate with others systems, when some data is needed. They must, now, appeal to this unique data source, becoming a non-mazy system.

The activities in this project are part of the scope of the DAS5511 – *Projeto de Fim de Curso* class, of the Control and Automation degree course of the Santa Catarina Federal University. The project was developed in Radix – Engineering and Software company,

In the document, it is first presented the company, an appeal to the project, as well its objectives and work methodology. After that, basic concepts are shown, with all the theory and basis fundamentation of the work. Finally, the solution is presented, detailing some system aspects, based on the theory presented before.

Keywords: monitoring, validation, automation.

Sumário

Lista de Figuras e Tabelas

Capítulo 1: Introdução

No século XIX, quando o primeiro poço de petróleo foi descoberto, pouco talvez se soubesse do quão importante seria esta descoberta para o futuro. A revolução industrial, ocorrida pouco depois, impulsionou o uso de energia para a manufatura e buscou-se várias alternativas para o carvão, a primeira fonte de energia das revoluções industriais.

O petróleo é uma fonte de energia não-renovável e seu interesse comercial cresceu vertiginosamente desde sua descoberta. Este foi, inclusive, a razão de crises e guerras, cujo vencedor buscava formas de dominar a exploração em certas áreas.

No Brasil, a primeira sondagem ocorreu ainda no final do século XIX, em São Paulo. Esta foi frustrada, tendo a perfuração apenas encontrado água sulfurosa. Apenas em 1939, na Bahia, é que se encontrou petróleo em terras brasileiras. Com o objetivo de monopolizar a exploração de petróleo no Brasil, a Petrobras – Petróleo Brasileiro S/A foi criada, em 1953 [2]. Hoje, diversas empresas atuam na exploração de petróleo e de energia, sendo a estatal brasileira uma das referências mundiais para exploração em águas profundas.

A exploração de petróleo e seu tratamento para ser um produto comerciável, criou diferentes ramos de atuação, com diversas empresas atuando em cada um desses lucrativos negócios: exploração, produção, refino, comercialização e transporte de petróleo e seus derivados.

O uso de petróleo e seus derivados é vasto. Além de gerar gasolina, combustível para motores de combustão interna, vários outros derivados do petróleo podem ser produzidos: parafina, gás natural, GLP, asfalto, nafta, querosene, solventes, óleos combustíveis, óleos lubrificantes, diesel e combustível de aviação.

A exploração do petróleo, comandada por uma única empresa, produzindo sua tecnologia, desde a exploração até o transporte é difícil. Assim, muitas empresas de prestação de serviço existem ao redor destas gigantes exploradoras de

petróleo. Desde a extração até o transporte, muitos serviços são terceirizados em busca de um produto final de qualidade.

Em 2007, uma enorme quantidade de petróleo em poços na camada de pré-sal foi descoberto no mar brasileiro. Devido a crise mundial que teve início em 2008, a exploração dos estimados 80 bilhões de barris de petróleo e gás no pré-sal não teve tanto investimento quanto esperado. Com o parecer do reestabelecimento da "normalidade" na economia mundial, a exploração desta área começou a receber grandes investimentos. Com o aumento da extração, as outras áreas de trabalho relacionadas com o petróleo, também estão sofrendo expansão.

Este trabalho, será desenvolvido no contexto do refino do petróleo desenvolvendo um sistema de validação e de um banco de dados para uma refinaria. A Radix - Engenharia e Software é responsável por produzir estes sistemas que irão gerir os dados da planta da refinaria, além de monitorar e validá-los.

1.1: Institucional

A Radix – Engenharia e Software, como pode ser visto na referência [3], foi fundada em abril de 2010, pelo ex-CEO da Chemtech, Luiz Rubião e outros ex-funcionários da Chemtech, que se tornaram sócios-diretores e sócios gerentes na Radix: Flávio Guimarães, João Chachamovitz, Alexander Cramer, Maurício Miele, Flávio Waltz e Paulo Rego. Outros funcionários foram recrutados, alguns da Chemtech (também empresa de engenharia e software) e, no início, menos de 50 funcionários estavam trabalhando na Radix. Sua sede está localizada no Rio de Janeiro – RJ, possuindo também um escritório em Belo Horizonte – MG.

Hoje, dois anos depois, a empresa conta com mais de 200 funcionários, altamente capacitados, e um faturamento maior que 20 milhões de reais. A missão da empresa é: "Oferecer serviços de engenharia e software diferenciados pela excelência técnica e ética e com independência tecnológica, ampliando os valores gerados para clientes, funcionários e sócios, a partir de um compromisso de longo prazo firmado com a sociedade, o País e o meio ambiente".

A empresa considera que um dos seus principais patrimônios é sua equipe. Isto é reconhecido pelos funcionários, haja visto que a Radix foi eleita a melhor empresa para se trabalhar no Brasil e a 4ª melhor da América Latina. Além disso, tem satisfação completa de seus clientes, contando também com certificações ISO.

Figura 1 - Áreas de Atuação da Radix

Na figura acima, estão as áreas de atuação da empresa. Como pode-se notar, há uma multidisciplinariedade nos campos apresentados, exigindo equipes também multidisciplinares e especializadas, capazes de atender as exigências dos seus clientes com agilidade e competência técnica. Alguns dos clientes da Radix, neste dois anos de empresa, são e foram: CSN, AkerSolutions, Keppel, TV Globo, Queiroz Galvão, Transpetro, Petrobras e Monsanto.

Além disso, a Radix considera essencial manter um relacionamento próximo com as instituições acadêmicas brasileiras, ajudando e patrocinando diversos eventos acadêmicos. Na UFSC, a empresa patrocina a equipe Vento Solar e em outras universidades patrocina também eventos como BAJA SAE e eventos de robótica.

1.2: Metodologia de Trabalho

Nesta seção, serão descritos como o trabalho se desenvolveu, o que foi estudado e o cronograma, além de como um engenheiro de controle e automação se situa na área. A Radix foi contratada para prestar serviços para uma refinaria e é neste contexto que o trabalho ocorreu, mais precisamente para a área de Transferência e Estocagem (TE) de uma refinaria.

1.2.1: Objeto de Estudo e Objetivos

Na refinaria, e como diversas indústrias, não se dispõe de uma ferramenta centralizada para armazenagem, monitoração e validação de dados, em tempo real. Isto reduz a confiabilidade dos dados disponibilizados pelas várias fontes de informação da planta, podendo comprometer informações geradas. Como não é possível classificar a qualidade dos dados, poucas ações podem ser realizadas no sentido de minimizar a propagação de erros. Além disso, existe a necessidade de uma padronização no tratamento e disponibilização das variáveis de processo (valor e qualidade) para outras aplicações.

Isto implica que os softwares da refinaria monitorem as entidades necessárias para automação e suporte operacional das áreas, obtendo dados diretamente da planta. Essa monitoração não é padronizada. Cada aplicação (isoladamente) é responsável pela validação de seus dados. A padronização e unificação das monitorações são dificultadas, pois existem diversos bancos de dados na refinaria.

Idealmente, as aplicações ou sistemas instalados devem utilizar um único Banco de Dados de Tempo Real como fonte de dados, a BDTR. A partir do desenvolvimento desta base centralizada, a monitoração e validação dos sinais serão padronizadas e unificadas.

O software de monitoração faz parte do conjunto de aplicações projetadas para auxiliar as operações da refinaria, se comunicando essencialmente com a BDTR. A solução permitirá: uma validação unificada e centralizada dos sinais (dados); a monitoração de outras aplicações (componentes e serviços); e o monitoramento dos estados da planta.

A partir deste acompanhamento e dos alertas emitidos pelo sistema (disponíveis para outras aplicações), problemas poderão ser identificados e solucionados com antecedência, aumentando a produtividade da área, a confiabilidade dos dados disponibilizados e minimizando a propagação de erros. Uma eficiência maior para o processo em geral será alcançada.

A base de dados (BDTR) receberá os dados da planta e de seus SDCD's (Sistema Digital de Controle Distribuído) através do protocolo OPC, portanto se configurando como um cliente OPC. O SDCD, por sua vez, colherá os dados da planta, e se comunicará, idealmente, única e exclusivamente com a BDTR.

Este sistema centralizado disponibilizará dados para os sistemas de níveis acima deste, também via protocolo OPC. Portanto, a BDTR também será um servidor OPC, atuando como cliente e servidor, neste protocolo. Assim, os sistemas de mais alto níveis não buscarão dados diretamente na planta (nos SDCD's), e sim passarão a ter a BDTR como fonte de dados.

O software de monitoração será um cliente OPC que consumirá dados da BDTR, executará diversos algoritmos para validação destes dados e os disponibilizará novamente na BDTR. Os diversos clientes da BDTR poderão obter estes dados validados ou os dados originais, colhidos da planta. A aplicação destes algoritmos, entretanto, garantem dados mais confiáveis, mais robustos e com menor propagação de erros para quem quiser consumí-los. Alguns destes algoritmos prevêem a exclusão de valores com erros grosseiros, aplicação de filtros e outras validações de sinais.

1.2.2: Desenvolvimento do Projeto

Para a obtenção de resultados conclusivos, as atividades serão desenvolvidas com supervisão do orientador do projeto na Radix, Diogo Pacheco, e também sob orientação do professor Dr. Rômulo Silva de Oliveira.

Além disso, diversoss documentos gerados para o desenvolvimento do projeto foram estudados. Assim, antes do início do desenvolvimento ocorreu a revisão bibliográfica para entendimento do problema e da solução proposta.

Estes documentos definem a função de um sistema de software e foram uma boa documentação que garante o desenvolvimento de um software de qualidade. São exemplos destes documentos: requisitos funcionais e não funcionais do projeto, arquitetura da solução, regras de negócio e casos de uso..

Após estudada a bibliografia, a execução da solução será executada. Isto envolve a programação do sistema em si, a partir das regras de negócio e casos de uso, respeitando a arquitetura projetada. A princípio, o projeto previa 12 meses (um ano) para sua conclusão. Entretanto, devido diversos fatores fizeram com que o prazo inicial fosse reestipulado.

Entretanto, para testar a viabilidade do projeto, uma parte mínima do sistema foi desenvolvida e entregue para testes na refinaria. Esta primeira parte envolve a validação de sinais da BDTR e a própria BDTR.

Com o seguimento do projeto, embora atrasados no prazo inicial, as demais funcionalidades serão agregadas e implementados na solução. Entre eles, sistemas integração de qualidade, monitoração de tanques de armazenamento e identificação de movimentação errônea, entre outros.

Por fim, a documentação necessária, tanto para documentação do projeto quanto para a disciplina DAS5511 – Projeto de Fim de Curso, é gerada. Entre os documentos estão: casos de teste para as funcionalidades implementadas (com resultados esperados, resultados do teste) para documentar a execução do projeto e relatórios executivos e monografia para a graduação.

1.2.3: O trabalho no contexto da Engenharia de Controle e Automação

Durante toda a graduação, disciplinas foram cursadas e podem ser aplicadas de maneira direta ou indireta ao projeto, algumas sendo vitais ao seguimento deste, outras apenas como auxiliares. Para iniciar citando disciplinas, este trabalho se desenvolveu no contexto da disciplina DAS5511 – Projeto de Fim de Curso. O professor Doutor Rômulo Silva de Oliveira participa como orientador acadêmico e como visto abaixo, ministra para o tripé de informática industrial da graduação.

Algumas das disciplinas que tiveram conteúdos aplicados de maneira direta no desenvolvimento global do projeto são as ligadas ao cerne de informática

industrial e foram fundamentais para a execução do projeto. Além disso, as disciplinas de automação foram importantes para o entendimento da arquitetura do planta e da escolha da melhor solução:

- DAS5305 e DAS5306 – Informática Industrial I e II: CLPs, arquiteturas de automação, programação concorrente, sistemas de tempo real.
- INE5225 – Fundamentos de Sistema de Banco de Dados: modelagem de dados, estruturas de armazenamento, requisitos funcionais.
- DAS5314 – Redes de Computadores para Automação Industrial: níveis hierárquicos no modelo CIM, redes industriais, projetos de padronização.
- DAS5315 – Sistemas Distribuídos para Automação Industrial: arquiteturas de sistemas distribuídos, CORBA, webservices.
- DAS5316 – Integração de Sistemas Corporativos: sistemas CIM; sistemas de informação e de armazenamento: banco de dados e web-servers; interoperação de sistemas: CORBA & DCOM; interoperações de dados: XML.

Outras disciplinas puderam ter conteúdos aproveitados, quando enfocadas para o contexto do projeto, e propiciaram a base para o desenvolvimento, como dito, principalmente as ligadas a informática industrial, que apresentaram os fundamentos de programação. Além das disciplinas citadas, também devem ser lembradas todas as que foram de pré-requisito para estas, haja visto que forneceram uma base sólida para a elaboração e execução do projeto.

Capítulo 2: Fundamentos para Solução

Neste capítulo são apresentadas as técnicas que embasam a solução, assim como a teoria que permite assegurar que a solução é viável e possível de ser implementada. Será apresentado primeiramente, como a comunicação deve ser feita, via OPC. Após isso, será apresentado como essa comunicação será utilizada e programada para assegurar que os requisitos funcionais sejam implementados.

2.1: OPC

Diversos protocolos existem hoje para controle e comunicação nas plantas industriais. Cada um destes protocolos possuem vantagens e desvantagens que, em baixo nível, em nível de planta, contribuem para um melhor desempenho global da operação. Entretanto, diversos padrões e protocolos de comunicação acabaram gerando sistemas de automação de grande complexidade, compostas por redes que pouco interoperam entre si.

Por cada protocolo ter também vantagens em relações a outros, a especificação da solução da planta industrial com produtos de um único fabricante é difícil. Além de ser virtualmente impossível em alguns casos, tal abordagem não é desejável do ponto de vista de mercado, pela dependência que se cria de um mesmo fornecedor.

Dada a necessidade, entretanto, de haver a troca de informações nas mais diversas camadas de aplicação como também de diferentes protocolos, um protocolo aberto e padronizado foi desenvolvido. O OPC (inicialmente denominada *OLE for Process Control*, baseada em tecnologia Microsoft, hoje, chamado de *Open Process Control*) surge neste sentido. Este protocolo foi desenvolvido por um grupo de fabricantes em conjunto com a Microsoft, com objetivo de possibilitar a troca transparente de dados entre diversos tipos de aplicações, integrando redes heterogêneas, no modelo Cliente/Servidor. Hoje, o protocolo OPC é mantido por diversas empresas e suas especificações podem ser encontradas no website da fundação OPC [4].

Um servidor OPC se conecta a um outro dispositivo como um Controlador Lógico Programável (CLP), Unidade de Terminal Remoto (RTU), Sistema Digital de Controle Distribuído (SDCD) ou outra fonte de dados como um banco de dados ou interface com usuário. Ao receber esta informação, o dado é transformado para o formato padronizado do OPC. A interoperabilidade é garantida via criação e manutenção de padrões abertos.

Dessa forma, um cliente OPC pode se conectar a este servidor e ler e escrever dados de e para dispositivios. Dispositivos clientes são, por exemplo, IHM (Interfaces Humano-Máquina), aplicações desenvolvidas ou um historiador de log. A padronização faz com que a necessidade de programar um cliente específico para cada dispositivo de campo não exista mais, com a troca de dados ocorrendo de maneira comum, com uma interface padrão e de maneira fácil de ser reutilizada.

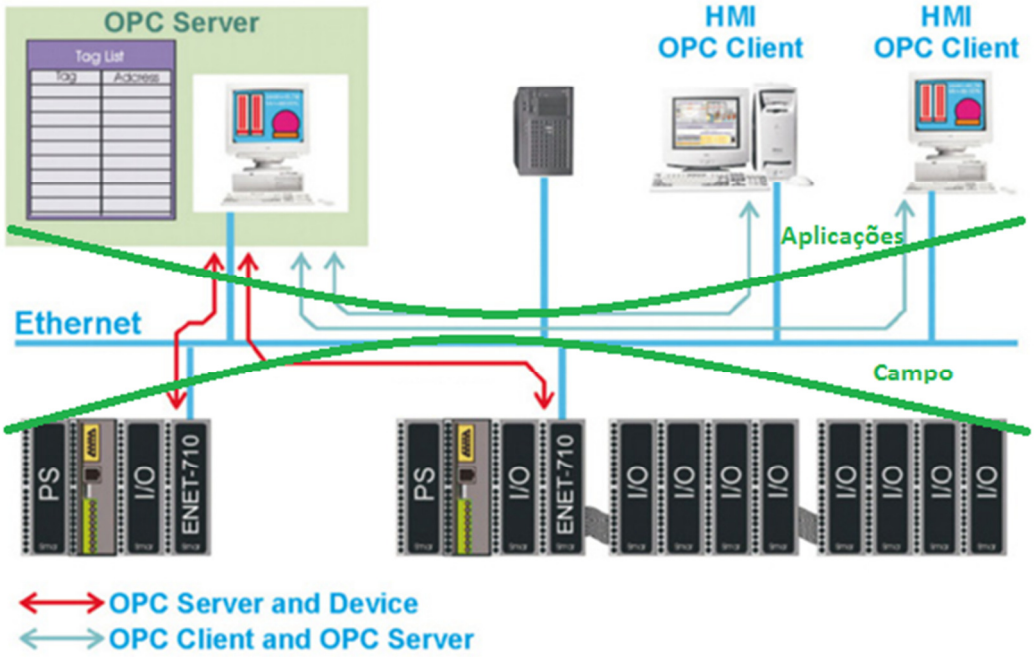


Figura 2 - Campo, Servidor OPC e Clientes OPC

A figura acima mostra o servidor OPC conectado a diferentes dispositivos no campo. Estes dipositivos podem ser de diferentes fabricantes e que se comunicam entre outros dispositivos de campo com seu protocolo proprietário. O servidor OPC recebe esses dados e disponibiliza a seus clientes de maneira padronizada. Por exemplo, se modificarmos o servidor OPC, de um desenvolvedor para outro, todo o sistema deve continuar a funcionar, sem maiores problemas.

Note, porém, que pode haver duas formas de comunicação: o SDCD ou o CLP é um provedor de dados OPC para aplicações clientes, sendo já um servidor OPC ou este provê um driver para que um servidor OPC obtenha os dados de suas entradas e saídas, com os valores de campo.

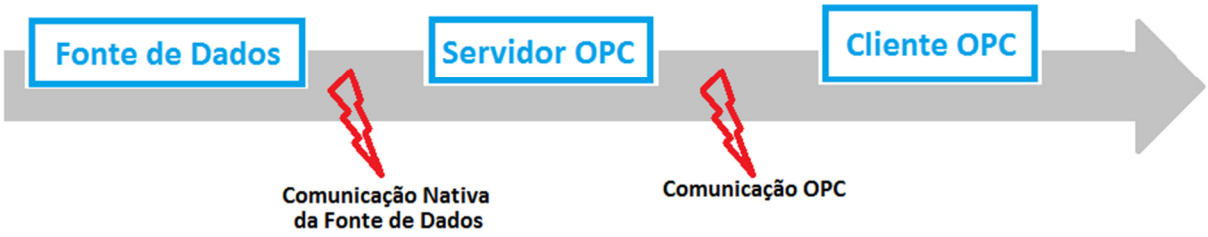


Figura 3 - Fluxo dos Dados OPC

Existem diversas especificações OPC atualmente, já desenvolvidas ou em desenvolvimento. As principais são OPC DA (*Data Access*) e OPC A&E (*Alarms & Events*) e que são as utilizadas neste trabalho. O OPC DA é a especificação original: usada para mover dados em tempo real de CLPs, SDCDs, e outros dispositivos de controle como IHMs e aplicações cliente. Atualmente, a versão 3.0 é a utilizada, permitindo alterar dados do sinal OPC, como qualidade do sinal e timestamp.

O OPC A&E disponibiliza alarmes e eventos sob demanda (em contraste com a troca de dados contínua do OPC DA). As informações trocadas neste protocolo são alarmes disparados pelo processo, ações do operador, mensagens adicionais e mensagens de acompanhamento e auditoria.

Outros protocolos que desenvolvidos ou estão em desenvolvimento são: OPC Batch, OPC Data eXchange, OPC Historical Data Access, OPC Security e OPC XML-DA. O OPC Unified Architecture (UA) é a próxima geração de padrões OPC que proverá uma plataforma coesa, segura e confiável para acessar dados e eventos históricos e em tempo real.

2.2: C# .NET

C# é uma linguagem de programação fortemente tipada, orientada a objetos, imperativa, declarativa e funcional [1]. Foi desenvolvida pela Microsoft dentro da plataforma .NET. A linguagem C# almeja ser simples, moderna, de propósito geral com orientação a objetos. A primeira solução buscada para a monitoração e

validação de sinais foi desenvolvida em C#. Houve algum progresso, mas nunca, de fato, chegou-se a concluir. A Microsoft mantém um website [7], com fóruns, discussões e documentações, que facilita o acesso a informações para desenvolvedores.

2.2.1: Framework Radix

O Framework Radix foi desenvolvido por funcionários da empresa com intuito de agilizar e tornar mais eficiente e padronizado o desenvolvimento de software. Ele é mantido também pelos funcionários e já foi aplicado a diversos projetos de diferentes clientes.

Um framework contém várias implementações que são corriqueiras nos sistemas. Obviamente, os sistemas são diferentes uns dos outros, mas pode se abstrair ações que são comuns, como acesso a dados, inserção, exclusão e edição (*CRUD – Create, Read, Update and Delete*). Esta é uma vantagem do framework que, se tomado para desenvolver do início, poderia levar meses para se atingir o objetivo.

O framework Radix está baseado no modelo MVC (*Model-View-Controller*), em que diferentes camadas são desenvolvidas, independentemente das outras. Assim, o sistema ganha escalabilidade, podendo cada uma das camadas ser executada em diferentes máquinas, por exemplo. Além disso, havendo a necessidade de retrabalho de código, com substituição de uma tecnologia por outra (por exemplo, de C# para Java), apenas a camada pode ser substituída, sendo transparente para as outras o que ocorre neste meio. Basta, para isso, editar as interfaces para que sejam ligadas novamente às diferentes camadas.

A camada de modelo (*Model*) é a que persiste e gerencia os dados e as estruturas de dados que serão utilizados pelo sistema. Representa a informação que a aplicação opera.

A camada de visão (*View*) exibe os dados para um operador ou transfere os dados para outra aplicação, sendo a saída do sistema. Pode haver mais de uma camada de visão desenvolvida para o sistema; uma em sistemas desktop, outra em smartphones e tablets e outra em ambiente web. Entretanto, todas tem a mesma

representação de dados (*Model*) e processam a informação da mesma maneira (*Controller*).

Como dito no parágrafo anterior, a camada de controlador (*Controller*) recebe as entradas externas (do operador ou de outro software), busca as informações necessárias da camada de modelo e realiza operações sobre estas informações, gerando uma nova informação para ser representada na camada de visão e persistindo novos dados na camada de modelo.

Diferentes alternativas para este modelo foram propostas e são utilizadas, principalmente no que se refere ao que cada um pode ou não fazer e com quais camadas se comunica. Em alguns modelos uma camada pode ser desmembrada em duas, mas, ainda, são baseadas no modelo MVC.

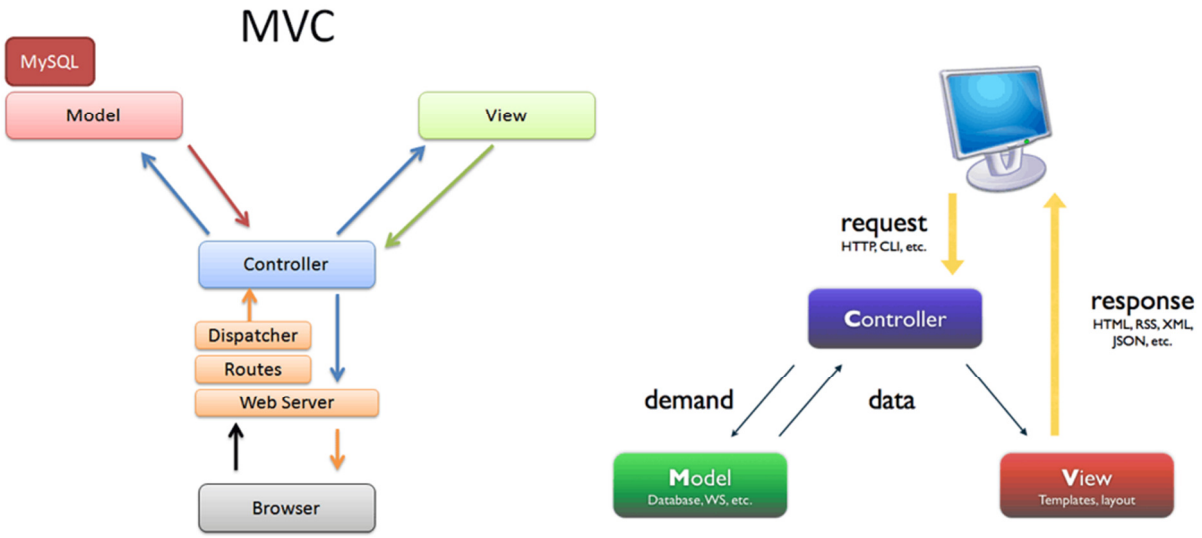


Figura 4 – Duas representações de modelo MVC

No desenvolvimento C#, com utilização do framework, as camadas MVC foram mantidas, sendo inseridas interfaces entre elas para fazer a ligação entre as camadas. Uma ou outra funcionalidade também foi transferida para a interface, mas a idéia geral é manter as camadas MVC e suas ligações.

2.2.2: Banco de dados

O sistema de banco de dados persiste as informações do sistema. Dessa forma, ao buscar uma informação necessária para a sua execução, esta é buscada no sistema de banco de dados.

O Sistema Gerenciador de Bancos de Dados (SGBD) utilizado neste projeto é o Microsoft SQL Server. Existem diversos SGBDs atualmente no mercado, sendo o Oracle o dominante. Estes sistemas gerenciadores armazenam, por vezes, dados de diversos bancos de dados e permitem que a informação desejada pelo sistema seja requisitada no momento que a informação deve ser exibida para o usuário. No modelo MVC, representa a camada de *Model*.

A chamada para busca de informações, pode ser feita dentro da linguagem de desenvolvimento, no caso C#. Definido em linguagem de marcação, a chamada ao sistema de banco de dados pode ser feita de maneira transparente no código, facilitando a programação e evitando que várias interfaces sejam implementadas em programas diferentes.

A necessidade de armazenar informações é evidente em qualquer sistema e o uso de SQL (*Structured Query Language*) é explicado por ser um grande padrão de banco de dados, além de ser simples e fácil de usar. O SQL foi baseado em álgebra relacional e permite que a forma do resultado seja especificado e não aénas caminho para chegar até ele. Seu uso foi difundido, por ser mais simples e mais confiável do que armazenar estas informações em arquivos na máquina e realizar buscas nestes.

2.3: Ferramenta Automatizadora de Procedimentos

Diversos ambientes de desenvolvimento para diversas linguagens de programação existem, atualmente, no mercado. Alguns tem finalidades mais específicas e estão intimamente ligadas a uma linguagem (como MQL e Metatrader, para operações de bolsa de valores), outros são genéricos e de propósito geral (C#).

Uma ferramenta desenvolvida em Lua, linguagem de script, é utilizada em algumas refinarias. Esta ferramenta tem algumas características que facilitam a operação e automação desses ambientes, como: escalabilidade; arquitetura

distribuída, no modelo cliente-servidor; ser de fácil entendimento, por possuir parte de sua programação visual; eficiente, por utilizar script Lua, um dos mais rápidos scripts existentes atualmente; extensibilidade, sendo o desenvolvedor capaz de incorporar novas funcionalidades de acordo com as diferentes necessidades dos ambientes de automação.

Esta ferramenta, com aplicação em refinarias, é utilizada para automação de procedimentos como partida e parada de equipamentos, execução de cálculos sequencialmente e repetidamente, tendo seu desempenho satisfatório em diversos casos. A comunicação OPC é utilizada por esta ferramenta para atuar em SDCDs e alterar e inserir valores em servidores OPC.

O desenvolvimento de funções e da estrutura de dados é feito em Lua, na etapa que chamaremos de “codificação”. Estando as diversas funcionalidades desenvolvidas (por exemplo, cálculos, automatismos, definições de dados), estas são importadas na ferramenta.

Assim, a lógica do programa fica disponível em blocos funcionais e operações de tomadas de decisão, que podem ser graficamente dispostos para realizar uma função, como se fossem um fluxograma.

O operador insere os parâmetros na estrutura de dados e endereços de TAGs OPC que serão lidos de um servidor OPC, durante a execução do fluxograma.

Essa etapa de configuração é feita em uma interface gráfica, que, na arquitetura cliente-servidor, é o cliente. Por fim, a operação e aplicação do que foi configurado é realizada por um servidor da ferramenta.

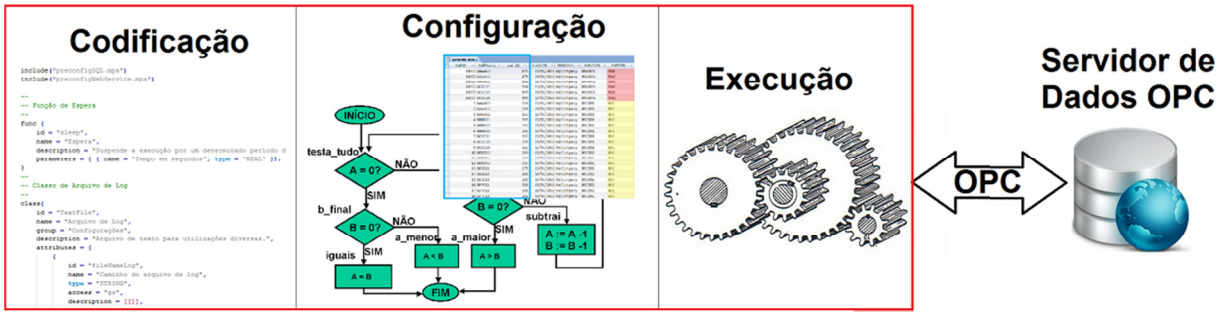


Figura 5 – Arquitetura da Ferramenta Automatizadora de Procedimentos

Este servidor, é um interpretador Lua, que com os códigos desenvolvidos na codificação e a configuração (que também é salva em arquivo Lua) realizará

operações de leitura e escrita OPC, além das ações programadas. Note, que este servidor que interpreta os códigos Lua é um cliente OPC para a operação.

2.3.1: Codificação em Lua

A primeira etapa no processo de utilização da ferramenta é a codificação. Esta consiste na definição da estrutura de dados (classes), além de funções de propósito geral. Tudo isto é desenvolvido em Lua, para posterior disponibilização no cliente da ferramenta, quando os códigos são importados, para a formação dos fluxos. A linguagem de script Lua possui um manual de referência online [8] que facilita o acesso de informações a programadores.

A estrutura de dados define atributos (variáveis ou parâmetros) e métodos. Os dados são separados em classes de operação, com cada classe tendo uma finalidade específica no sistema. Os atributos identificam um ponto de controle que será utilizado no processo, isto é, um ponto OPC ou então características inseridas pelo operador, referentes as classes de operação. Os métodos realizam operações sobre os atributos destas classes, obtendo e salvando valores nos itens OPC e realizando operações, conforme dados dos parâmetros inseridos pelo operador.

As funções de propósito geral, são funções que serão disponibilizadas para a geração dos fluxogramas. Tantos estas quanto os métodos serão executadas no fluxo para a obtenção do fim desejado.

2.3.2: Geração de Fluxogramas e Definição de Atributos de Classes

Desenvolvidos os códigos, temos que realizar duas configurações na aplicação cliente da ferramenta: configuração das classes de operação e configuração dos fluxogramas. As funções desenvolvidas na codificação são disponibilizadas em blocos funcionais, que são organizados nos fluxos. As classes de operação tem seus atributos configurados e mapeados com os respectivos pontos OPC. Então, podem ser feitos os fluxogramas que serão executados.

A configuração das classes de operações consiste na especificação dos sinais e itens OPC que serão utilizados para a execução e parâmetros de execução. Diferentes classes, em diferentes quantidades e com diferentes finalidades podem

ser criados. Os atributos disponíveis para um classe de operação é definido na codificação.

Os pontos OPC representam um ponto lógico, no servidor OPC (podendo ser um equipamento físico, como sensores). Existem quatro tipos básicos de pontos: booleano, real, inteiro e textual. Estes pontos de controle precisam ter referências válidas para que a execução não cause erros na execução. Portanto, estes itens devem ser criados no servidor OPC (note que os tipos básicos são os mesmos permitidos no padrão OPC).

A configuração dos fluxogramas determinam a execução dos procedimentos. A linguagem visual de fluxograma permite um entendimento mais simples do que em código escrito, o que pode ser uma vantagem para programadores iniciantes. Note, entretanto, a necessidade de ter os blocos funcionais que comporão o fluxo previamente definidos na codificação. Assim, o fluxo estabelece a sequência de ações e decisões a serem executados durante o procedimento.

2.3.3: Servidor de Execução

O Servidor de Execução, como dito, possui um interpretador Lua, que, de acordo com os fluxogramas, faz as chamadas de funções desenvolvidas na codificação. Além disso, é este servidor que se comunica com o processo, via OPC, sendo, portanto, um cliente OPC. Este é o elemento central da ferramenta, que, de fato, executa as operações sobre as classes de operação cadastradas, de maneira automatizada, interagindo com um servidor OPC para manipular seus pontos OPC.

A arquitetura do sistema permite que o servidor seja executado em uma máquina diferente da aplicação de configuração. Além disso, diversos servidores de execução (podendo ser em apenas uma máquina) podem se comunicar com diferentes servidores OPC. Isto provê uma flexibilidade ao sistema. O servidor de execução carrega as informações de classes de operação e funções de propósito geral (da codificação), os dados configurados e fluxos (do ambiente de configuração) e interpreta os fluxogramas fazendo as chamadas de métodos das classes de operação.

Capítulo 3: Desenvolvimento

Neste capítulo será apresentado o que foi desenvolvido para obter a solução, englobando todo o contexto do projeto. Partiremos da solução do servidor OPC e sua conexão com o SDCD, até chegarmos as soluções propostas para a monitoração e validação de sinais.

É importante ressaltar, entretanto, que o autor deste relatório não foi o responsável pelo desenvolvimento da BDTR (seção 3.1). Esta foi desenvolvida em paralelo com a monitoração. O desenvovimento da BDTR foi apenas acompanhado pelo autor, dando suporte, quando necessário, auxiliando e realizando testes desta.

A monitoração e validação de sinais (seção 3.2) foi a parte do projeto em que o autor realmente esteve envolvido e, de fato, implementou. Tanto as soluções para a monitoração em C# .NET, quanto na ferramenta automatizadora de procedimentos são apresentadas. Os algoritmos validam e monitoram sinais analógicos, digitais e multiestado.

É importante ressaltar a supervisão e orientação tanto de Diogo Pacheco, orientador na empresa e de Dr. Rômulo S. de Oliveira, orientador acadêmico, que auxiliaram todo este desenvolvimento quanto a documentação do processo.

Outros dois estagiários estiveram envolvidos em soluções que comporão, também, a solução final, com mais ações incorporadas a monitoração de dados, como cálculos referentes a tanques.

3.1: BDTR

O processo de refino possui diversos sistemas divididos por funcionalidades e que precisam obter dados a todo instante e com garantia de recebimento. Dessa forma, a solução de uma Base de Dados em Tempo Real (BDTR) para o processo visa suprir a necessidade de dados, executando as tarefas solicitadas pelos sistemas da refinaria em um tempo rígido e eficiente.

Apesar da existência de vários padrões para comunicação, a manutenção das informações precisa ser otimizada, rápida e confiável. Os dados das ordens e

movimentações contidos no banco de dados de transferência e estocagem precisam ser acessados por diversos sistemas, além de outros precisarem acessar os pontos diretamente no servidor OPC do SDCD. Dessa forma, um sistema que possui uma base de dados que, ao mesmo tempo, centraliza e garante a troca de informações em tempo real foi proposto.

Dessa forma, a BDTR será composta por um Servidor de dados OPC composto por OPC DA e A&E, de modo que clientes OPCs, aplicativos de banco de dados e Web Services sejam capazes de acessá-la e auxiliem as operações da refinaria. Assim, a BDTR deve prover e receber dados de e para qualquer software de suporte à automação que venha a necessitar de dados de processo.

O Servidor OPC da BDTR proverá dados às aplicações de suporte a automação da refinaria. Como o protocolo OPC segue normas padrão, qualquer outro servidor OPC poderia prover estes dados. Porém o desenvolvimento permite que funcionais adicionais as da especificação sejam adicionadas. Assim, conexões com alguns softwares já existentes (caso do software de otimização de misturas e movimentações) são providas. Além disso, acesso a outros bancos de dados da refinaria e emissão de alarmes via WebService são providos na solução da BDTR, funcionalidades não previstas na norma. Um toolkit (da Softing GmbH) foi utilizado no desenvolvimento, provendo um ponto de partida para a aplicação.

3.1.1: Servidor OPC DA

O Servidor OPC DA (*Data Access*) é quem mantém os dados [5]. Para cada dado é mantido um rótulo (TAG). Os dados mantidos podem ser de quatro tipos: real, inteiro, booleano ou textutal. Além disso, o dado possui consigo dois outros atributos além do valor: qualidade e estampa de tempo (*timestamp*).

O tipo do dado é configurado pelo engenheiro de TAG na criação da TAG. No escopo deste projeto, também foi desenvolvido uma interface em Excel, com programação em Visual Basic, para a criação e configuração de TAGs. Assim, cada TAG será configurada com informações como endereço da TAG, endereço de conexão com SDCD, valores mínimos e valores máximos, entre outros. Após finalizada a configuração, esta é exportada para a BDTR na própria interface.

A exportação usa um pipe nomeado. Este pipe nomeado é um canal que é criado entre dois processos e facilita a comunicação entre estes. Por exemplo, ao invés de exportar para um arquivo externo e, posteriormente, importar na BDTR, o pipe nomeado faz a “transferência” do arquivo diretamente, por este canal, podendo o canal ser fechado após o seu uso.

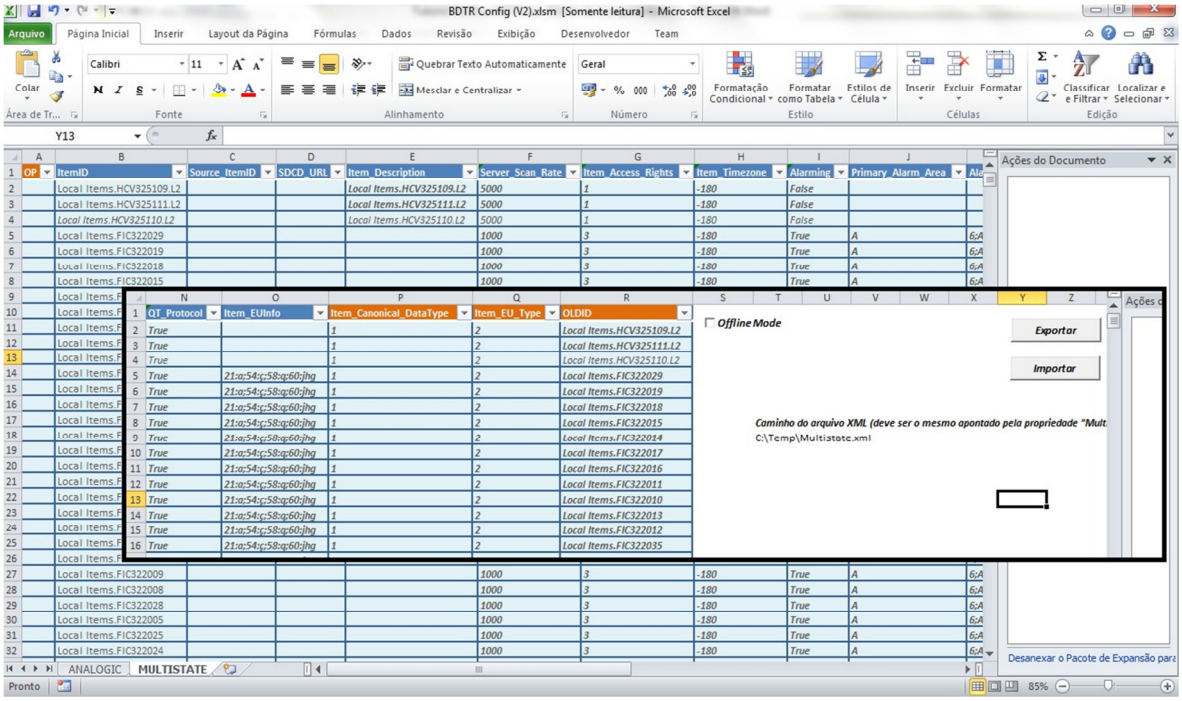


Figura 6 – Interface Excel da BDTR

Um aspecto importante da BDTR é a conexão com SDCD. A conexão com SDCD é realizada para itens da BDTR que possuem sinais ligados à planta. Podem ser sinais de nível, vazão, temperatura, entre outros. Entretanto, nem todos os itens cadastrados na BDTR possuem uma conexão com um SDCD. Alguns podem apenas estar presentes na BDTR e, por exemplo, ser resultado de cálculos realizados por outros softwares que é exatamente o princípio da validação de sinal, como mostrado na Figura 14 - Software de Monitoração e da BDTR unificada.

Note que a BDTR prevê que o SDCD seja também um servidor OPC, recorrendo a seus dados por chamadas OPC. O SDCD, no entanto, converte dados de campo para o padrão OPC. Estes dados podem ser Modbus e Profibus e a conversão para o padrão OPC consiste na inclusão de qualidade e timestamp, por exemplo, além do meio físico de comunicação ser diferente.

Além disso, é através da conexão com o SDCD que a atuação no campo ocorrerá e dados de campo poderão ser fornecidos. Como a BDTR prevê que múltiplos clientes podem estar conectados a ela ao mesmo tempo, a escrita para valores de campo podem ter diversas fontes e a BDTR atua apenas como transmissor da informação.

A informação de qualidade e de estampa de tempo é essencial para outros sistemas saberem se a informação que é fornecida a eles é útil e pode-se fazer uso daquele dado. As principais qualidades são: “boa”, “incerta” e “ruim”. A informação de qualidade será de fundamental importância para a validação de sinais, que é o foco deste trabalho. A espeficição 3.00 é a que permitiu que clientes OPC escrevessem em qualidade e timestamp nos servidores OPC.

Os dados referentes aos TAGs são mantidos na memória da BDTR e retransmitidos aos sistemas conforme sua necessidade. Entretanto, para iniciar sua execução, a configuração dos TAGs é mantida não em um banco de dados mas em arquivos XML, que mantém tudo o que é necessário para a criação de determinada TAG. Este arquivo também pode ser exportado da interface Excel. Assim, a BDTR já está apta a manter dados para todos os sistemas.

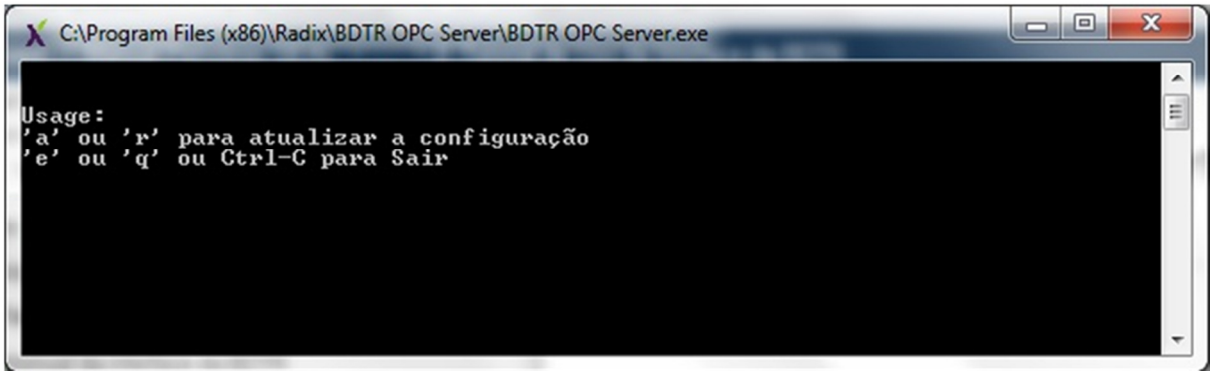


Figura 7 – BDTR executando

Os clientes podem adquirir dados através de três tipos de chamada, basicamente: síncrona, assíncrona e por subscrição.

A chamada síncrona é uma chamada bloqueante para o cliente. Este solicita o dado para a BDTR e não executa nenhuma ação até que a resposta com o dado seja provido. A chamada assíncrona é não-bloqueante para o cliente, permitindo que outras execuções sejam efetuadas enquanto a resposta da BDTR não ocorra.

A última, por subscrição, permite que o cliente faça uma “assinatura” junto a BDTR. Assim, de tempos em tempos, havendo uma mudança no valor junto a BDTR uma mensagem com o valor é enviado ao cliente que tem sua subscrição ativa (*dataChange*). Se nenhuma alteração ocorre no intervalo até a próxima mensagem ser enviada, uma mensagem de *keepAlive* é encaminhada para o cliente, apenas para que o cliente saiba que o servidor ainda está em operação.

A vantagem da subscrição é que diminui significativamente o tráfego de dados na rede e também não mantém o cliente bloqueado aguardando um dado. Entretanto, exige que o cliente seja capaz de receber e tratar este dado que pode chegar, a princípio, a qualquer instante.

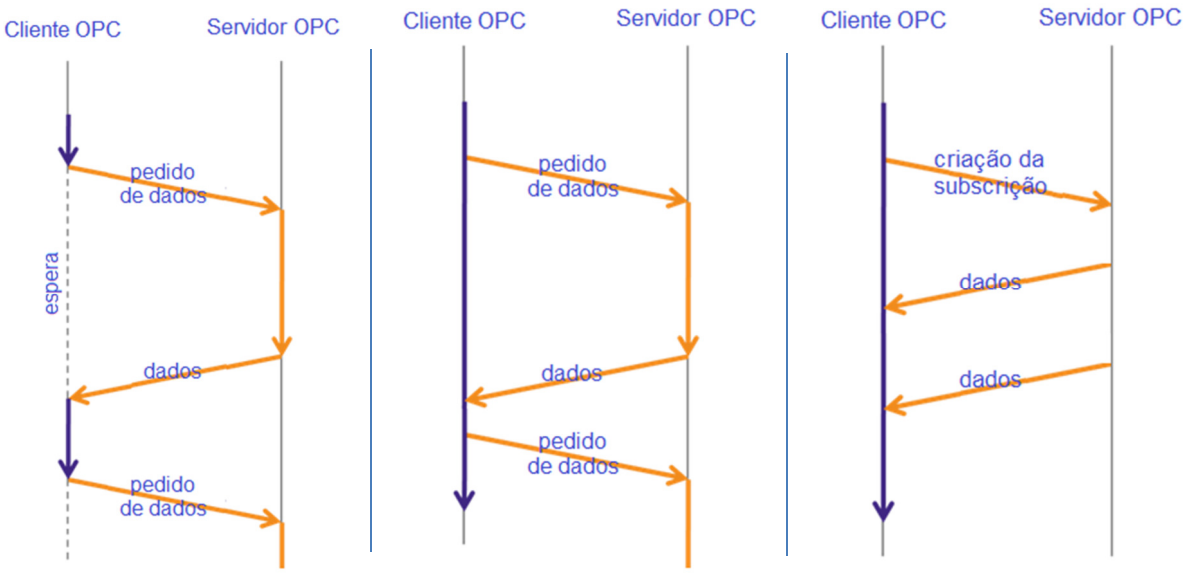


Figura 8 – Chamadas Síncrona, Assíncrona e por Subscrição, respectivamente

3.1.2: Servidor OPC A&E

O servidor OPC A&E (*Alarms and Events*) monitorará acontecimentos e condições sobre TAGs armazenadas na BDTR [6]. Eventos são mensagens enviadas aos operadores para que este tenha ciência de que algo ocorreu, entretanto, sem que este tenha que intervir e informar ao sistema que recebeu a mensagem. Alguns eventos, por exemplo, são: “A TAG XXXXX alterou seu valor de AAA para BBB” ou “A TAG YYYYY foi criada”. O sistema continua funcionando sem nenhum problema por estas mensagens serem perdidas.

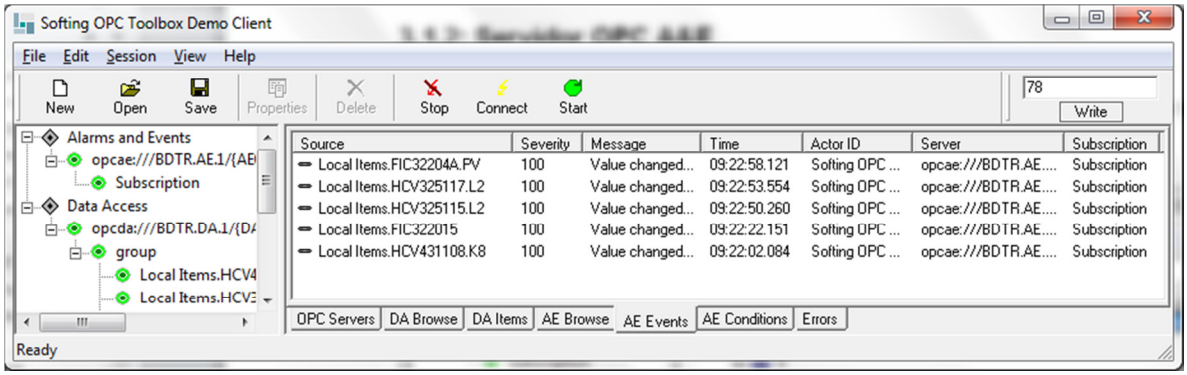


Figura 9 - Eventos disparados pela BDTR

Alarmes, entretanto, precisam que o operador reconheça seu estado, pois estas são situações mais críticas. Alarmes, normalmente, são emitidos quando TAGs estão fora de seus valores normais de operação, atingindo valores baixos (LO), muito baixos (LO-LO) ou altos (HI) e muito altos (HI-HI). Assim, o operador, ao verificar a condição anormal do sistema, pode tomar providências para assegurar que o sistema volte ao seu funcionamento adequado e, assim, marcar o alarme como reconhecido. Caso o alarme seja reconhecido, mas a condição anormal do sistema ainda ocorre (como um valor fora do normal) o alarme continuará ativo.

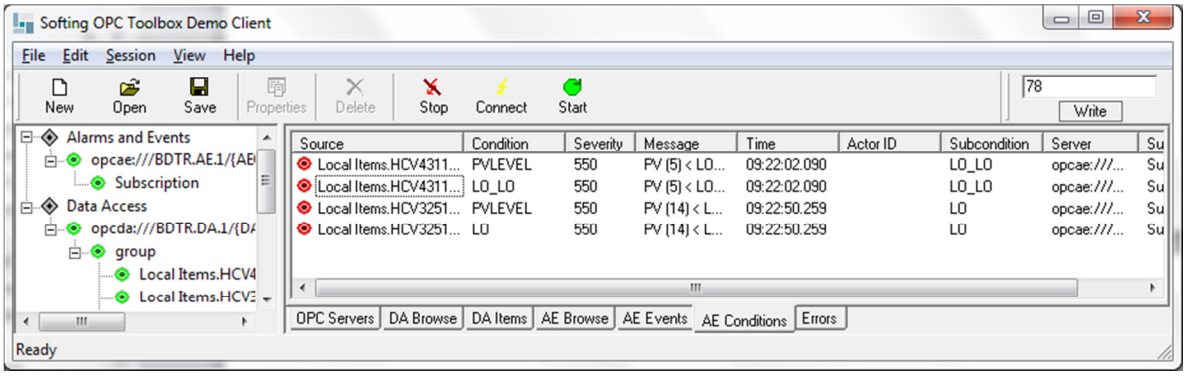


Figura 10 - Alarmes emitidos pela BDTR, sem reconhecimento

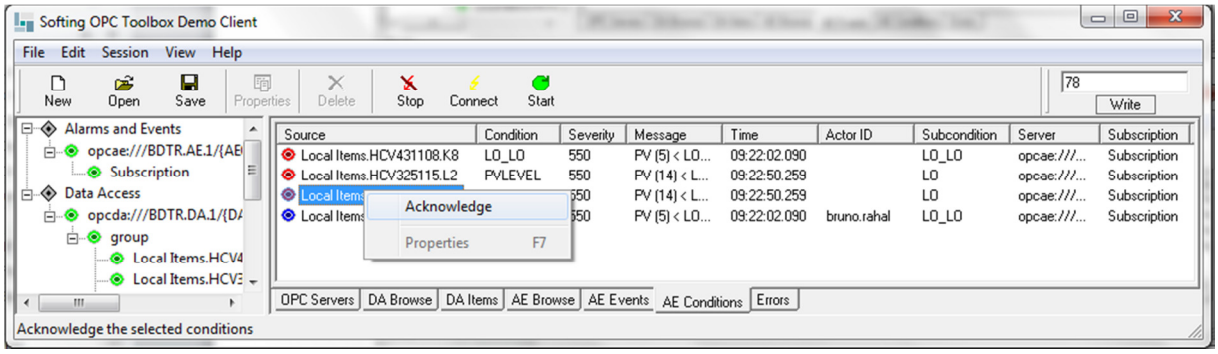


Figura 11 - Alarmes emitidos pela BDTR, sendo reconhecidos

Note, entretanto, que mesmo que uma condição anormal de funcionamento ocorra e volte ao normal no curso normal de operação do sistema, um alarme ainda estará presente e aguardando reconhecimento. Isto porque a condição anormal precisa ser conhecida para que possíveis providências sejam tomadas. Por fim, seu reconhecimento eliminará o alarme.

Além disso, é importante reparar que tanto alarmes quanto eventos tem uma severidade associada (que pode assumir valores inteiros de 1 a 1000) que pode ser utilizada para filtros e rastreabilidade dentro do sistema.

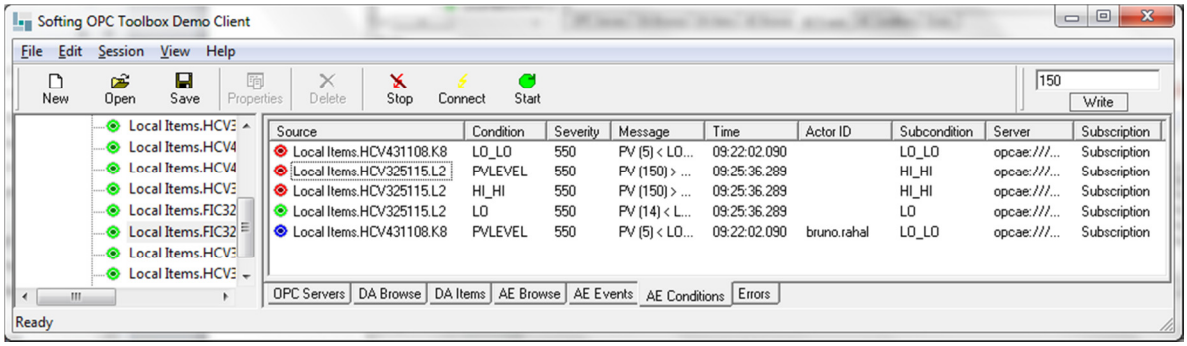


Figura 12 - Alarmes emitidos pela BDTR

Outro aspecto a ser apresentado da emissão de alarmes e eventos é sua divisão por espaços de área e espaços de eventos. O espaço de área se refere a definição OPC de área e fonte. O espaço de eventos se refere a definição OPC de categoria.

Todo alarme e evento emitido tem associado uma fonte a ele, que é quem causou este disparo. Esta fonte pode ou não ter uma ou mais áreas associada a ela. Uma área é, tipicamente, um agrupamento dos equipamentos da planta.

Também, todo alarme e evento emitido tem associado uma categoria a ele. Este conceito se refere ao tipo de disparo que foi efetuado. Por exemplo, podem ser disparos de categorias padrões como “Device Failure” ou “System Message” ou ainda outras categorias podem ser criadas como “Monitoração Planta”.

Um determinado operador pode estar interessado em receber eventos e alarmes de apenas uma área, que é a área que ele atua. Assim, diversas áreas podem ser especificadas. Outro operador, entretanto, pode estar interessado em

alarmes e eventos referentes a apenas algum processo, recebendo apenas mensagens sobre este processo, como Monitoração da Planta, na Figura 13.

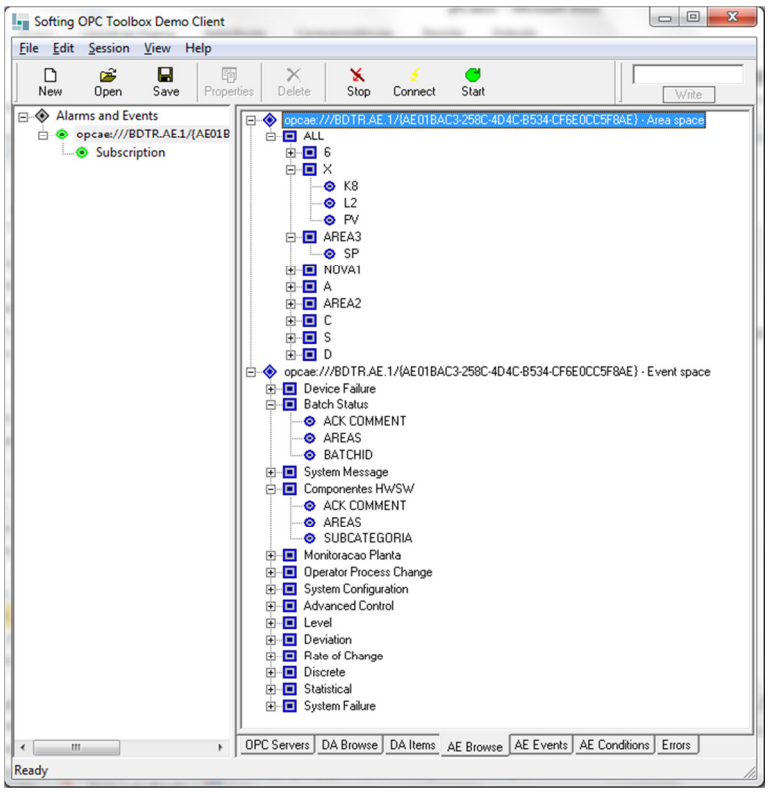


Figura 13 – Divisão por espaços de área e de eventos do servidor OPC A&E

Um alarme e/ou evento emitido sempre vai ter associado a ele uma categoria e uma fonte. Uma ou mais áreas podem ser especificadas.

Entretanto, a BDTR só está preparada para efetuar alguns tipos de alarmes e eventos, como extrapolação e mudança de valores. Há casos em que se deseja efetuar disparos por outros softwares, como é o caso da monitoração e validação de sinais, foco deste trabalho.

A primeira solução desenvolvida para atender estes disparos por chamadas externas foi com o uso de banco de dados. Todos as inserções que ocorressem no banco de dados eram lidas, disparadas e, posteriormente, apagadas. Assim, definia-se os parâmetros no banco de dados como fonte, categoria, área(s), severidade. Ao ler os valores, a BDTR efetuava o disparo, como se fosse originado por ele mesmo.

Entretanto, embora tenha funcionado, outra solução foi de fato mantida na solução final: uso de WebService. Assim, os softwares que necessitam realizar um disparo de alarme e/ou evento fazem a chamada direta à BDTR, que possue esta

interface para chamada de aplicações externas. Na chamada, são passados os mesmos parâmetros que eram armazenados no banco de dados e o disparo é efetuado.

3.2: Monitoração e Validação de Sinais

O sistema de monitoração e validação de sinais atuará sobre dados mantidos na BDTR. Estes dados passarão por cálculos e verificações de modo a garantir dados mais confiáveis para que outras aplicações possam realizar suas operações de maneira mais segura e eficaz. Cálculos sobre TAGs de valores analógicos, digitais e multiestado são realizados. A este sistema, ainda serão incorporados diversos outros algoritmos e métodos que ainda precisam ser formaçozadps e implementados. Alguns dos métodos que são escopo da continuação do projeto são: balanço de massa, monitorar estados de alinhamento, integração de qualidade e cálculos de atributos de tanques.

As regras criadas para validar os dados, são, em sua maioria, para que aspectos empíricos de operação de campo sejam formalizados e aplicados de maneira sistemática. Estas regras são descritas no documento de Regras de Negócio, que se refere a um documento de levantamento de requisitos do sistema.

Um exemplo desse empiritismo trazido a prática: dada uma tubulação, temos uma válvula com controle digital (aberta ou fechada) e após essa válvula, há um medidor de fluxo, de valores analógicos. Assim, considere que estamos interessados no valor do fluxo. Existe um algoritmo (chamado de TAG Condicional) que verificará o valor da válvula: aberta ou fechada. Se esta tiver aberta, o valor lido para o fluxo pode estar válido. Entretanto, se a válvula estiver fechada, qualquer valor lido para o fluxo é errôneo e, muito provavelmente, proviente de ruídos.

Obviamente, um valor para o fluxo sempre será lido e não faz sentido que seja escrito, forçadamente, um valor que consideramos válido no valor lido do sensor. Aliás, espera-se, neste caso, que o valor do fluxo seja configurado na sua configuração como somente leitura, pois não faz sentido alterar um valor de um sensor. Entretanto, pode haver casos (como o da válvula) que podemos ler e escrever. O software de monitoração não faz isso, pois seu foco não é a atuação e sim prover informações para softwares de suporte a automação tomar decisões e

atuar no sistema. Pode ocorrer, ainda, que alguns softwares não queiram os dados validados e sim os dados originais, sem nenhuma alteração da validação.

Dessa forma, para que seja mantido a informação original e ainda provermos uma validação do sinal que é lido do campo, uma TAG OPC, mantida também na BDTR, adjacente a original, será criada. Para diferenciá-las, um sufixo será acrescentado a endereço textual de cada TAG validada, com a base da TAG original. A figura 14 apresenta a estrutura geral de como a BDTR e o sistema de monitoração irão interagir. Esta estrutura de automação e como os sistemas interagirão estão descritas nos Documentos de Visão dos respectivos sistemas que apresentam requisitos funcionais e não funcionais que um software deve possuir.

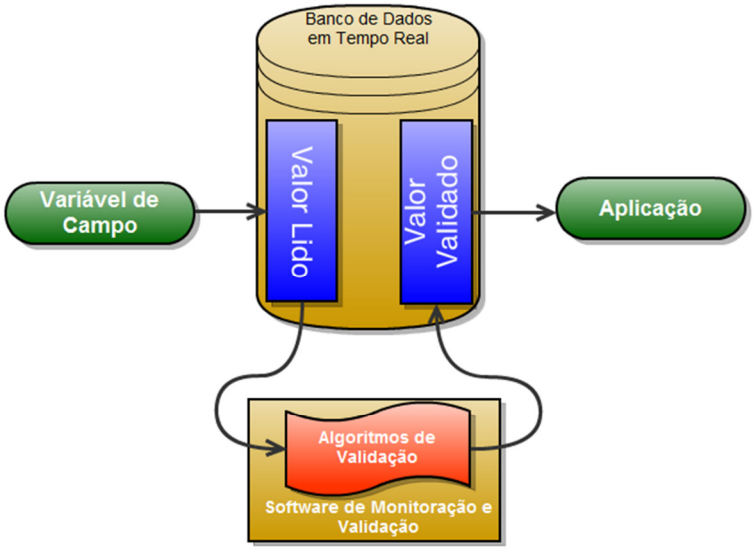


Figura 14 - Software de Monitoração e da BDTR unificada

Os tipos de valores armazenados na BDTR, como previamente apresentados, são: digital (booleano), analógico (real), multiestado (inteiro) e textual (string). Os algoritmos de validação aplicados em um tipo de TAG não serão aplicados em outro tipo. Não parece fazer o menor sentido verificar se o valor 13 é verdadeiro. Por isso, cada tipo de TAG terá o seu conjunto de algoritmos específicos.

Como dito, a idéia é que estes algoritmos formalizem algum tipo de empiritismo da operação, também respeitando o tipo das variáveis em que estes são aplicados.

3.2.1: Algoritmos de Validação

Os algoritmos de validação, como dito no tópico anterior, separam-se por tipo da TAG. Estes estão apresentados na tabela abaixo.

Os algoritmos foram levantados de acordo com especificações do sistema e estão bem descritos nos diversos documentos levantados na descrição do sistema, como o de Documento de Visão, Regra de Negócios e Casos de Uso. Estes documentos são parte da Engenharia de Requisitos do software, responsável por especificar os detalhes do software que toda boa prática de desenvolvimento deve realizar.

A leitura dos dados e sua escrita na BDTR e demais funções auxiliares são padronizadas e comuns, independente do tipo de TAG. Assim, a diferenciação de um tipo de TAG para outro se dá basicamente pelos seus algoritmos.

Analógica	Digital	Multiestado	Textual
Mínima Variação	Congelamento	Congelamento	NÃO SE APLICA
TAG Condicional	TAG Condicional	TAG Condicional	
Limite Inferior	Taxa de Variação Digital	Taxa de Variação de Estado	
Limite Superior	Filtro Digital	Filtro de Estado	
Taxa de Variação Máxima			
Média Móvel com Desvio Padrão			
Filtro de Sinal			

Tabela 1 – Tipos de TAG e seus algoritmos

Um aspecto importante é a frequência de validação que define o intervalo entre duas validações subsequentes. Esta frequência (em segundos) deve ser maior que 60 (um minuto). Muitos algoritmos utilizarão os valores antigos (validados

anteriormente), os valores lidos (que estão em campo), com suas estampas de tempo e intervalos de validação para realizar cálculos.

Note que os algoritmos são aplicados em sequência e um valor lido pode sofrer alteração em um algoritmo e ser entrada para outro algoritmo. Além disso, não é aplicado nenhum algoritmo para o tipo textual, uma *string*. A seguir, será descrito de maneira sucinta o que cada algoritmo foi especificado a realizar.

3.2.1.1: Algoritmos de Validação de TAG Analógica

Os algoritmos para TAGs analógicas realizam cálculos e verificações sobre valores do tipo real. Estes podem assumir, em teoria, qualquer valor no espectro de menos infinitio a mais infinito. Sabemos, entretanto, que cada equipamento possui suas limitações e valores esperados e isto é configurado na BDTR. Assim, os valores são limitados e basta, para a monitoração, a aplicação dos algoritmos.

O algoritmo de Mínima Variação atua no sentido de validar o sinal que sofre variação acima de uma banda morta. Assim, serão válidos os sinais que, entre uma validação e outra, sofreram algum tipo de incremento ou decremento acima do valor configurado de banda morta.

O algoritmo de TAG Condicional verifica se existe alguma condição, proveniente de uma TAG condicionante (e não a que está sendo validada), para que o valor que está sendo validado seja ignorado. Se tal condição ocorrer, nenhum outro algoritmo é aplicado e o valor lido é atribuído ao valor validado.

Os algoritmos de Limite Inferior e Limite Superior tornam válidos os valores que estejam compreendidos entre os valores configurados para estes limites.

O algoritmo de Taxa de Variação Máxima verifica se a diferença entre dois valores dividida pelo intervalo entre duas validações extrapola um valor determinado, validando o sinal que não extrapolar.

O algoritmo de Média Móvel com Desvio Padrão mantém uma janela de dados com valores, qualidade e estampas de tempo e calcula o desvio padrão destes valores armazenados. Se o desvio padrão calculado for menor que um valor de referência, a média móvel entre os valores é calculada e este será o valor do sinal validado.

O algoritmo de Filtro de Sinal aplica um filtro digital de primeira ordem, com base no valor lido e no último valor validado.

Note que em caso das condições verificadas tornarem os sinais inválidos, valores de restrição pré-definidos pelo operador ou pelo algoritmo serão atribuídos ao valor atribuído. Entretanto, a qualidade deste sinal deixará de ser uma qualidade BOA e será considerada INCERTA ou ainda RUIM.

3.2.1.2: Algoritmos de Validação de TAG Digital

Os algoritmos para TAGs digitais realizam cálculos e verificações sobre valores do tipo booleano. Estes podem assumir verdadeiros ou falsos, ligados ou desligados, abertos ou fechados, dependendo do equipamento.

O algoritmo de Congelamento torna válido o sinal que cuja diferença entre os timestamps dos sinais validado anteriormente e lido é inferior ou igual ao tempo de congelamento, parâmetro configurado. Assim, este algoritmo busca identificar TAGs que devem ser atualizadas e não são, podendo haver erros na transmissão dos sensores, por exemplo. Poderia-se pensar como o análogo à Mínima Variação para a TAG Analógica, mas, como a TAG Digital só pode possuir valores discretos, o algoritmo de Congelamento foi especificado.

O algoritmo de TAG Condicional tem a mesma idéia de aplicação que para TAGs analógicas, porém o valor atribuído à TAG validada é digital.

O algoritmo de Taxa de Variação Digital pretende não manter o mesmo valor para um sinal por intervalos muito longos. Assim, os valores são lidos e sendo do mesmo valor, um contador é incrementado com a diferença de tempo entre duas validações subsequentes. Estando este contador acima de um patamar configurado, o sinal é invalidado, sendo válido apenas quando este troca de valor ou enquanto estiver abaixo do intervalo de tempo configurado. Neste último caso, o sinal validado recebe o valor do sinal lido.

O algoritmo de Filtro Digital faz a mesma verificação que o algoritmo de taxa de variação digital. Entretanto, estando este valor abaixo do patamar de tempo configurado para este algoritmo, nada é realizado. Se o valor é extrapolado, a TAG validada recebe o valor lido.

Note que em caso das condições verificadas tornarem os sinais inválidos, valores de restrição pré-definidos pelo operador ou pelo algoritmo serão atribuídos ao valor atribuído. Entretanto, a qualidade deste sinal deixará de ser uma qualidade BOA e será considerada INCERTA ou ainda RUIM.

3.2.1.3: Algoritmos de Validação de TAG Multiestado

Os algoritmos para TAGs multiestado realizam cálculos e verificações sobre valores do tipo inteiro. Estes valores podem ter associações textuais, como, por exemplo, para um TAG o valor 13 aceitar o valor “aberto” e o 28 “fechado”. Entretanto, o valor para o TAG mantido na BDTR é o valor numérico. Essa associação textual-valor é feita pela BDTR que converte o valor textual para o inteiro.

O algoritmo de Congelamento tem a mesma idéia de aplicação que para TAGs digitais. O algoritmo de TAG Condicional tem a mesma idéia de aplicação para TAGs analógicas, porém o valor atribuído à TAG validada é multiestado.

Já os algoritmos de Taxa de Variação de Estado e Filtro de Estado são análogos aos algoritmos de Taxa de Variação Digital e Filtro Digital, porém, valores inteiros são representados. Assim como os outros algoritmos, um sinal inválido tem valores de restrição aplicados e qualidade alterados de acordo com o algoritmo aplicado.

Na verdade, na implementação da BDTR, a TAG digital é apenas uma TAG multiestado com dois estados. Entretanto, para a aplicação dos algoritmos sua diferenciação foi necessária para adequação ao que era esperado no levantamento do sistema, como nas regras de negócio.

3.3: Monitoração em C#

A monitoração em C# .NET foi a primeira solução proposta e chegou a ser desenvolvida por um curto período, até ser deixada de lado para a solução com a ferramenta automatizadora de procedimentos. Entretanto, esta solução não está totalmente descartada para o produto final, em que funções serão incorporadas a monitoração e validação de sinais. Esta última é o foco desta dissertação,.

O desenvolvimento utilizando a plataforma C# .NET foi facilitado pelo uso do framework da Radix, já citado neste documento. O modelo MVC é o utilizado neste framework.

3.3.1: Modelo de Dados

Para a criação do modelo de dados, foi usado o Enterprise Architect, uma ferramenta utilizada para documentação de projetos. Também foi criado neste software o diagrama de casos de uso, diagrama de classes e outros tipos de documentação. A vantagem deste software é gerar o script SQL para a geração do banco de dados automaticamente a partir do modelo de dados. O Microsoft SQL Server 2008 é o banco de dados utilizado na solução apresentada.

O modelo de dados mantém a configuração da validação de sinais. Esta configuração é mantida no banco de dados e é descrita pelo caso de uso específico (na figura abaixo, “Inserir, Visualizar, Editar, Excluir Validação de Sinal”). O engenheiro de TAGs ou engenheiro processista gerará a configuração de acordo com o funcionamento adequado esperado para a operação. O sistema, utilizando-se dessa configuração, fará a validação do sinal em si (“Monitorar e Validar Sinal”), lendo os valores do servidor OPC e reescrevendo valores. Abaixo, estão o diagrama de casos de uso e modelo de dados para o sistema de monitoração e validação de sinais.

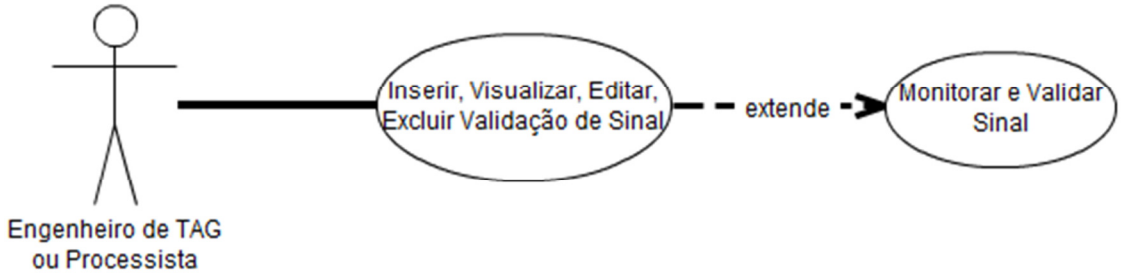


Figura 15 - Diagrama de Caso de Uso - Validação de Sinal

Para a monitoração e validação de sinais, duas tabelas são necessárias: uma que mantém as TAGs OPC (que se relacionarão com diversas outras tabelas, quando novas funções forem incorporadas) e uma que mantém a configuração de cada algoritmo da monitoração. Assim, a chave primária da tabela das TAGs OPC

será chave estrangeira para diversas outras tabelas, inclusive a de monitoração e validação de sinais.

Na tabela de configuração, cada algoritmo terá seus parâmetros para execução. Serão valores como flags (booleanos identificando a execução ou não de cada algoritmo), inteiros (tipo de critérios e identificadores), strings (endereços) e reais (valores).

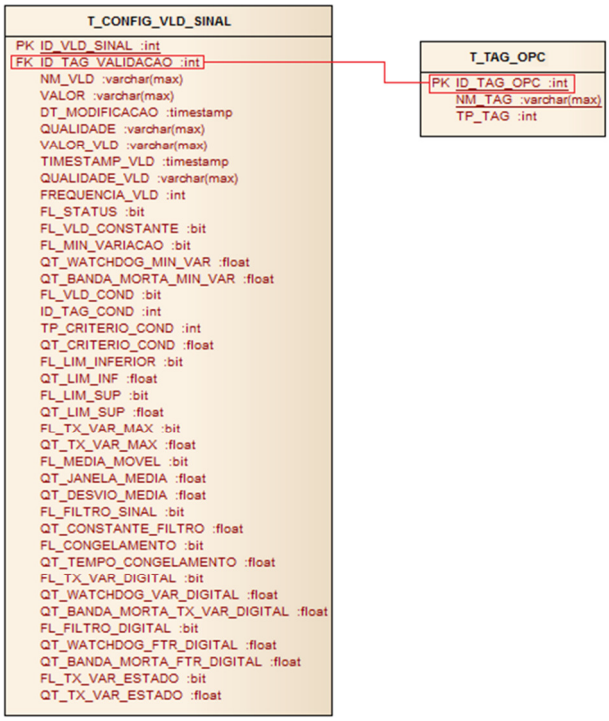


Figura 16 – Modelo de Dados - Validação de Sinal

As entradas da tabela de configuração se remetem aos algoritmos de validação descrito na seção 3.2.1:. Outras mantêm no banco de dados o valor e o valor validado das TAGs OPC que serão lidos do servidor OPC. Estes são armazenados no banco de dados para futura apresentação ao usuário, assim como a qualidade e o timestamp.

Nesta camada, cada tipo de TAG terá uma classe de modelo em C# que fará a vinculação do banco de dados com a instância da configuração armazenada nele. Assim, estas configurações estarão disponíveis para a aplicação. Dessa forma, tem-se o que é necessário para realizar a validação.

Do modelo MVC, o *Model* é apresentado como descrito acima. Restam ser descritos *View* e *Controller* que serão apresentados a seguir.

3.3.2: Interface

Nesta solução, para o usuário manter e inserir a configuração dos sinais, foi escolhida uma interface Web. Esta consiste na camada *View* do modelo MVC. Além de ser a camada que apresentará as configurações e permitirá ao usuário inserir novas configurações, o valor dos sinais original e validados também serão exibidos.

Primeiramente, a interface foi desenvolvida para testes e verificar a funcionalidade do sistema. Um webdesigner fez, posteriormente, uma versão melhorada do que seria o sistema. Entretanto, o sistema foi deixado de lado para a utilização da ferramenta automatizadora de procedimentos, como dito.

A implementação das interfaces se deu diretamente em código HTML, utilizando CSS para alteração do estilo. Scripts para interatividade com o usuário também foram necessários, como Javascript. Após a criação dos elementos na tela, bastou relacionar os elementos com itens da programação em C#. O framework faz todo o trabalho de fazer esta ligação, sendo necessário apenas passar como parâmetro o nome do elemento no código.

Abaixo, estão algumas imagens da interface desenvolvidas.

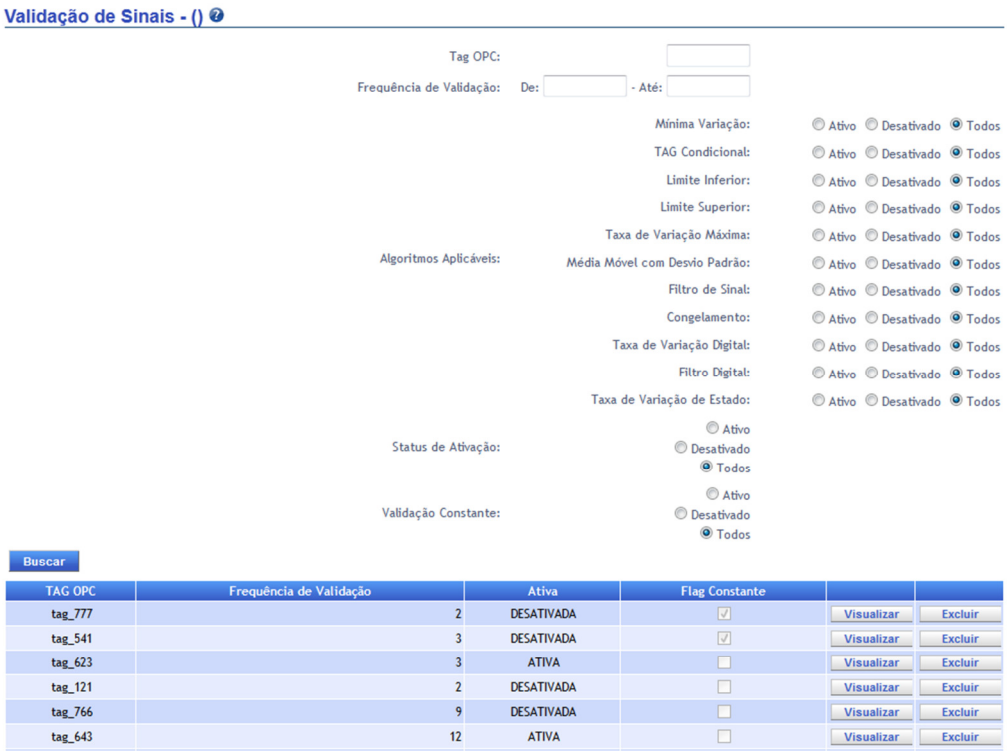


Figura 17 - Filtragem de Configurações

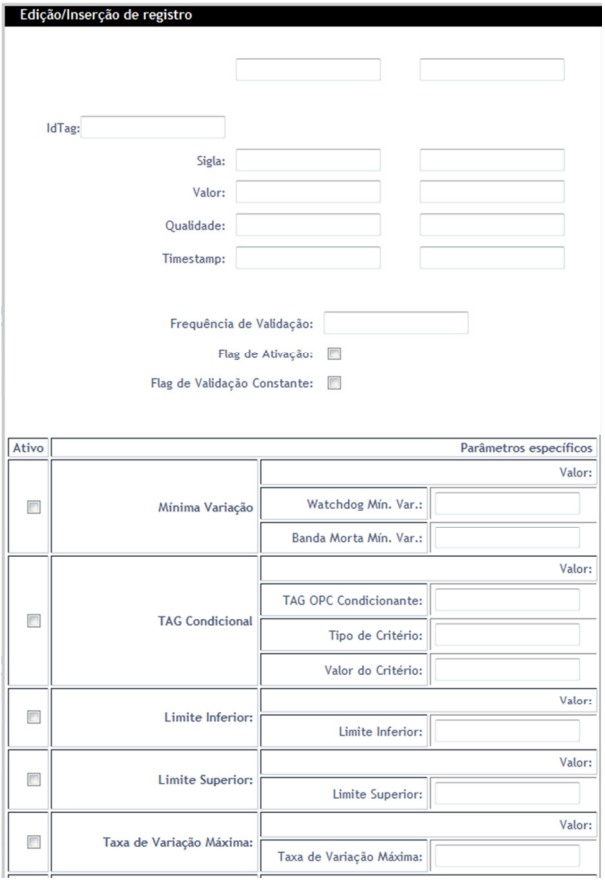


Figura 18 - Criação de Configuração Analógica

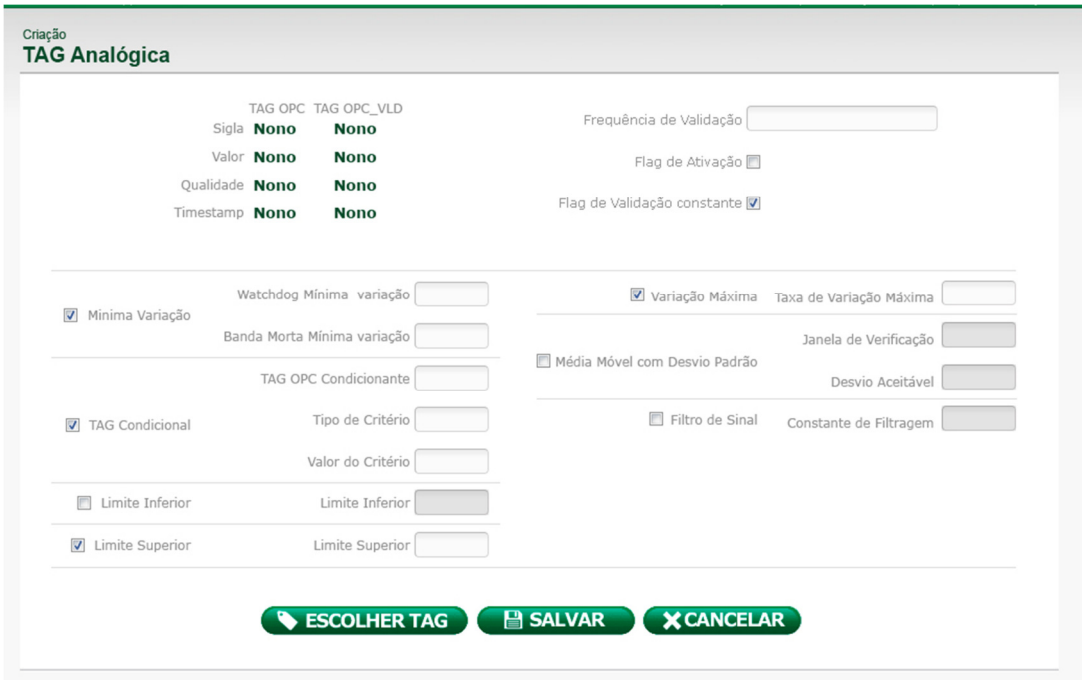


Figura 19 - Criação de Configuração Analógica Feita pelo Webdesigner

Note que os elementos apresentados na tela também se remetem aos algoritmos de validação apresentados na seção 3.2.1:. Estes items serão armazenados no banco de dados, apresentado na seção 3.3.1: Modelo de Dados.

Por exemplo, ao inserir valores para a Mínima Variação, estando a *check box* selecionada, Watchdog Min. Var. com valor 10 e Banda Morta Min. Var. com valor 2 uma inserção na tabela "T_CONFIG_VLD_SINAL" ocorrerá. Uma nova linha será inserida com os valores para os parâmetros "FL_MIN_VARIACAO" = *true*, "QT_WATCHDOG_MIN_VAR" = 10, "QT_BANDA_MORA_MIN_VAR" = 2.

Entretanto, diversos outros valores deveriam ser inseridos por não poderem ser nulos na tabela, como o "ID_TAG_VALIDACAO". Outros valores têm valores limitados, como "FREQUENCIA_VLD" deve ser maior que 60. Todas estas limitações poderiam ser trabalhadas após a inserção, retornando um erro se não estivessem adequadas. Porém, haveria um processamento extra por parte do servidor. O uso de scripts poderiam alertar o usuário destas condições e a inserção enviada para o servidor já estar de acordo com uma configuração correta.

A camada *View* é, por fim, aquela em que o usuário interage com o sistema. Apresenta os valores que estão mantidos no banco de dados e estão também no servidor OPC.

3.3.3: Implementação das Regras de Negócio

Na camada de *Controller* é onde o processamento da validação acontece. As Regras de Negócio definem o que foi apresentado na seção 3.2.1 e, nesta camada são codificadas, para posterior compilação e execução adequada do sistema.

Abaixo, vamos exemplificar a Regra de Negócio para o algoritmo de Mínima Variação:

Algoritmo de Validação (AV) – Mínima Variação

Para calcular a mínima variação, são necessários os valores do sinal anteriormente validado e do sinal original. Estes valores são comparados com o valor de banda morta, para a mínima variação, seguindo a seguinte fórmula: |Valor do sinal anteriormente validado – valor do sinal| > banda morta.

Algoritmo de Validação – Mínima Variação – Banda morta maior que a diferença dos valores dos sinais

Se a diferença absoluta, dos valores dos sinais validado e original, for superior a banda morta, o valor original do sinal é válido. Este valor é então atribuído ao valor do sinal validado e o temporizador referente a este AV é reinicializado.

Algoritmo de Validação – Mínima Variação – Banda morta menor ou igual que a diferença dos valores dos sinais

Se a diferença absoluta, dos valores dos sinais validado e original, for superior a banda morta, um temporizador é incrementado. Este temporizador recebe o valor de tempo, em minutos, da diferença entre duas execuções subsequentes do AV – Mínima Variação.

Algoritmo de Validação – Mínima Variação – Temporizador maior que tempo associado ao Watchdog

Quando o temporizador possuir valor superior ao tempo associado ao Watchdog o sinal é inválido. O valor do sinal validado recebe o valor conforme definido nas opções de saída de um AV, definidos na Regra de Negócio – Classificar Saída Inválida.

Algoritmo de Validação – Mínima Variação – Saída Inválida e Valor de Restrição

Caso a opção de saída, definido pela Regra de Negócio – Classificar Saída Inválida, seja o valor de restrição, ao valor do sinal validado é atribuído um valor igual do sinal original somado ou subtraído da banda morta, de modo que este valor esteja compreendido entre os valores lidos.

Algoritmo de Validação – Mínima Variação – Saída Inválida e Qualidade de Restrição

Caso a opção de saída, definido pela Regra de Negócio – Classificar Saída Inválida, seja o valor de restrição, a qualidade do sinal segue a restrição de sub-status de qualidade limitada inferior, quando ao sinal original foi adicionado o valor de banda morta, ou superior, quando ao sinal original foi subtraído o valor de banda morta.

Acima, estão descritas diversas regras de negócio que compõem o que será usado para validar o algoritmo de mínima variação. Note que pode haver referência para outras regras de negócio (Regra de Negócio – Classificar Saída Inválida, não apresentada). Em uma codificação "português estruturado", por exemplo, podemos apresentar o algoritmo de Mínima Variação da seguinte maneira:

Algoritmo de Validação – Mínima Variação

O algoritmo de mínima variação recebe os seguintes parâmetros:

- Watch dog: valor em minutos;
- Banda morta: valor em unidade de engenharia (UE).

A execução é composta pelos seguintes passos:

1. Ler o valor da TAG Original e anterior (TAG validada);
2. Calcular a diferença (atual - anterior), comparando com a banda morta (absoluta);
3. Se a diferença (modulo) for superior à banda morta então:
    a. Sinal do TAG válido;
    b. Zerar o contador;
    c. TAG Validada recebe valor atual.
4. Senão incrementa o contador com o intervalo entre validações para a TAG;
5. Se o contador for superior ao watch dog então:
    a. Sinal do TAG inválido;
    b. TAG Validada recebe o valor conforme definido nas opções de saída de um AV;
    c. Caso a opção de saída seja o valor de restrição, deve ser igual ao valor atual (anterior) somado (subtraído) com a banda morta. Atente que este valor deve estar entre os valores lidos e deve seguir a restrição de sub-status de qualidade (limitada inferior ou superior).
6. Se o contador for superior ao watch dog então:

Note que esta não é a definição da Regra de Negócio para o algoritmo de Mínima Variação, por não possuir diversas características que uma Regra de

Negócio deveria possuir: atomicidade, independência tecnológica, não fazer menção a implementação. Entretanto, nos dá um bom ponto de partida do que deve ser implementado.

```
private void MinimalVariation()
{
    if (qualityUnderValidation != (Int32)Softing.OPCToolbox.EnumQuality.BAD)
    {
        if (Math.Abs(Convert.ToDouble(validation.Value) - Convert.ToDouble(validation.ValueVLD)) > validation.MinVarDeadband)
        {
            timer = 0;
        }
        else
        {
            timer += Math.Abs(validation.DateSignal.Ticks - validation.DateSignalVLD.Ticks);
            if (TimeSpan.FromTicks(timer).TotalMinutes > validation.MinVarWatchDog)
            {
                isValid = false;
                qualityUnderValidation = (Int32)Softing.OPCToolbox.EnumQuality.UNCERTAIN;
                if (validation.MinVarAVPossibleResult == 1 || validation.MinVarAVPossibleResult == 2)
                {
                    valueUnderValidation = Convert.ToDouble(validation.MinVarAVResultValue);
                }
                else if (validation.MinVarAVPossibleResult == 3)
                {

                    if (Convert.ToDouble(validation.Value) - Convert.ToDouble(validation.ValueVLD) > 0)
                    {
                        qualityUnderValidation = qualityUnderValidation + (Int32)Softing.OPCToolbox.EnumQualityLimit.LOW;
                        valueUnderValidation = Convert.ToDouble(validation.ValueVLD) + validation.MinVarDeadband;
                    }
                    else
                    {
                        qualityUnderValidation = qualityUnderValidation + (Int32)Softing.OPCToolbox.EnumQualityLimit.HIGH;
                        valueUnderValidation = Convert.ToDouble(validation.ValueVLD) - validation.MinVarDeadband;
                    }
                    else if (validation.MinVarAVPossibleResult == 4)
                    {
                        qualityUnderValidation = (Int32)Softing.OPCToolbox.EnumQuality.BAD;
                        valueUnderValidation = Convert.ToDouble(validation.Value);
                    }
                }
            }
        }
    }
}
```

Figura 20 – Implementação da Mínima Variação em C#

Acima, está o código do algoritmo de Mínima Variação em C#, para a camada de Controller. Este é apenas um dos algoritmos. Embora a parte do sistema da validação de sinais, em si, estivesse bem encaminhada no desenvolvimento em C#, ainda falta a codificação que conectaria este sistema com o servidor OPC.

Assim, quando o projeto se encaminhava para o desenvolvimento deste “conector”, a idéia de continuarmos com a implementação em C# foi abandonada e partimos para uma solução já existente, que já era capaz de fazer essa conexão com o servidor OPC e também capaz de ter implementados os algoritmos de validação: a ferramenta automatizadora de procedimentos.

3.4: Monitoração na Ferramenta Automatizadora de Procedimentos

A Monitoração e Validação de Sinais desenvolvida na ferramenta automatizadora de procedimentos foi a solução que foi implementada que teve seu funcionamento testado. Algumas vantagens já foram apresentadas previamente e pode-se citar: a ferramenta foi desenvolvida especificamente para uso em refinarias e aplicações ligadas ao petróleo. Outra vantagem é a comunicação OPC já estar implementada e incorporada na ferramenta, já que esta é a forma que a ferramenta se comunica com a planta. Assim, basta realizar o desenvolvimento das classes de operação, a codificação e os fluxogramas.

3.4.1: Codificação

Assim como a camada de *Controller* no modelo MVC, é na codificação onde são mantidas as regras de negócio do sistema. Entretanto, a coordenação lógica das chamadas das funções, no caso do que foi desenvolvido, não são mantidas por código (como seria em C#). A coordenação lógica é mantida pelos fluxogramas, sendo cada função do código disponibilizado em um bloco funcional no fluxograma.

Fazendo a analogia com o modelo MVC, a camada de *Model* é análoga, em partes, com a Configuração, mantendo toda a configuração que será instanciada (no Fluxograma, durante a execução) e utilizada nos cálculos de validação de sinais. Na codificação também são criadas as estruturas de classes de operação e seus atributos, que serão instanciadas.

Abaixo, vemos a implementação do algoritmo de validação para Mínima Variação. A estrutura para emissão de alarmes e eventos para a BDTR A&E também é mantida na codificação. A mensagem é montada e a chamada “alerta:alertar()” faz com que um evento seja emitido pela BDTR A&E.

```
methods = {
        {
            id          = "av_minima_variacao",
            name        = "AV - Mínima Variação",
            description = [[AV - Mínima Variação.]],
            parameters  = { },
            results     = { },
            code =
                function(self)
                    if self.qualidadeSobValidacao > 32 then -- if self.qualidadeSobValidacao > 32, pra considerar todos os tipos de BAD
                        if ((self.valorSobValidacao - self.valorVLD) > self.min_var_deadband) then
                            self.timerMinVar = 0
                        else
                            if self.timerMinVar == nil then
                                self.timerMinVar = math.abs(self.timestampSobValidacao - self.timestampVLD)
                            else
                                self.timerMinVar = self.timerMinVar + math.abs(self.timestampSobValidacao - self.timestampVLD)
                            end
```

```
                    if (self.timerMinVar > self.min_var_watchdog) then
                        self.ehValido = false
                        self.qualidadeSobValidacao = 64 --INCERTA

                        mensagem = os.date("%c").." TAG "..self.tag.." Invalidando TAG! Algoritmo de Mínima Variação: o valor de Watchdog:
                        "..self.min_var_watchdog.." e o ultimo valor que satisfazia o deadband foi a "..self.timerMinVar.." segundos atrás"
                        alerta.mensagem = mensagem
                        alerta.fonte = self.tag
                        alerta:alertar()
                        local ll = fh:write(mensagem.."\n")

                        if (self.flag_saida_padrao_min_var == 1) then
                            self.valorSobValidacao = self.valor_saida_padrao_min_var
                        elseif (self.flag_saida_padrao_min_var == 2) then
                            self.valorSobValidacao = self.ultimoValorBom
                            self.timestampSobValidacao = self.ultimoTimestampBom
                        elseif(self.flag_saida_padrao_min_var == 3) then
                            if (self.valorSobValidacao > self.valorVLD) then
                                self.qualidadeSobValidacao = 65 -- UNCERTAIN + LOW LIMITED
                                self.valorSobValidacao = self.valorVLD + self.min_var_deadband
                            else
                                self.qualidadeSobValidacao = 66 -- UNCERTAIN + HIGH LIMITED
                                self.valorSobValidacao = self.valorVLD - self.min_var_deadband
                            end
                        elseif(self.flag_saida_padrao_min_var == 4) then
                            self.qualidadeSobValidacao = 0      --BAD
                            self.valorSobValidacao = self.val
                        end
                    end
                end
            end
        end
    },
}
```

Figura 21 - Implementação da Mínima Variação em Lua

Assim, cada função implementada aqui estará apta a ser chamada no Fluxograma. Além disso, diversos atributos foram criados para estarem disponíveis na configuração.

3.4.2: Configuração da Planta

A estrutura das classes de operação, criada na codificação, deve ser preenchida na configuração pelo engenheiro de processo ou engenheiro de TAG. Assim, todos os parâmetros que estavam no modelo de dados foram criados nas classes.

Além disso, há uma TAG de Origem de uma TAG de Destino que são, respectivamente, as que serão lidas para serem validadas e as validadas. Cada algoritmo representa uma classe de operação diferente. Cada validação de sinal terá uma área específica para que seus atributos tenham valor, conforme a operação desejada.

Assim, note que as *strings* TAG Origem e TAG Destino são os endereços da BDTR DA e devem possuir exatamente o mesmo texto desta.

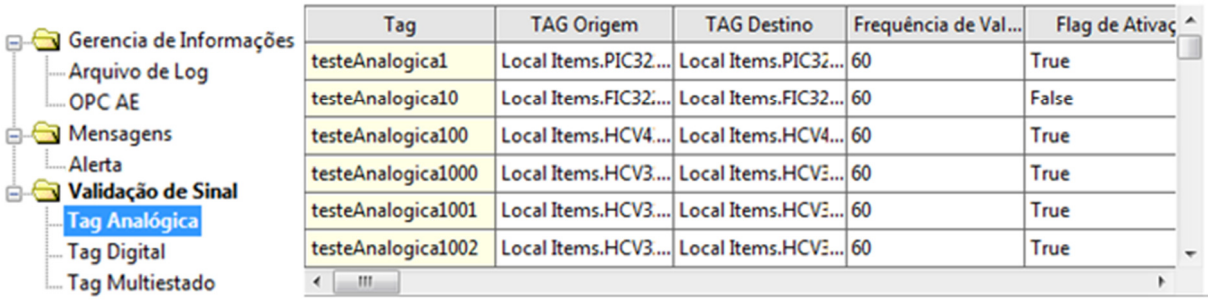


Figura 22 – Configuração e Inserção de Parâmetros para TAGs Analógicas

Figura 23 – Resumo das configurações inseridas

Estando todas as configurações prontas, os algoritmos podem ser criados nos fluxogramas.

3.4.3: Geração dos Fluxogramas

Os fluxogramas mantêm a coordenação lógica da execução. Abaixo, estão os fluxos para as validações das TAGs Analógica e TAGs Digital. Novamente, nos remetemos as regras de negócio para saber quais algoritmos devem ser executados e qual a ordem de execução.

Assim, cada fluxo executará em um *loop* infinito, após serem instanciados. A estrutura é de fácil entendimento por ser extremamente parecido com um flograma. Os losangos representam pontos de decisão, retornando a comparação verdadeira ou falsa. Os retângulos são pontos de ação, que executam funções definidas na codificação. Note o algoritmo de Mínima Variação no fluxo de TAG Analógica.

Os fluxos de TAG Digital e TAG Multiestado são muito semelhantes, como pode ser visto pelas descrições das regras de negócio, já na seção 3.2.1. Assim, a figura do fluxo de TAG Multiestado foi suprimida.

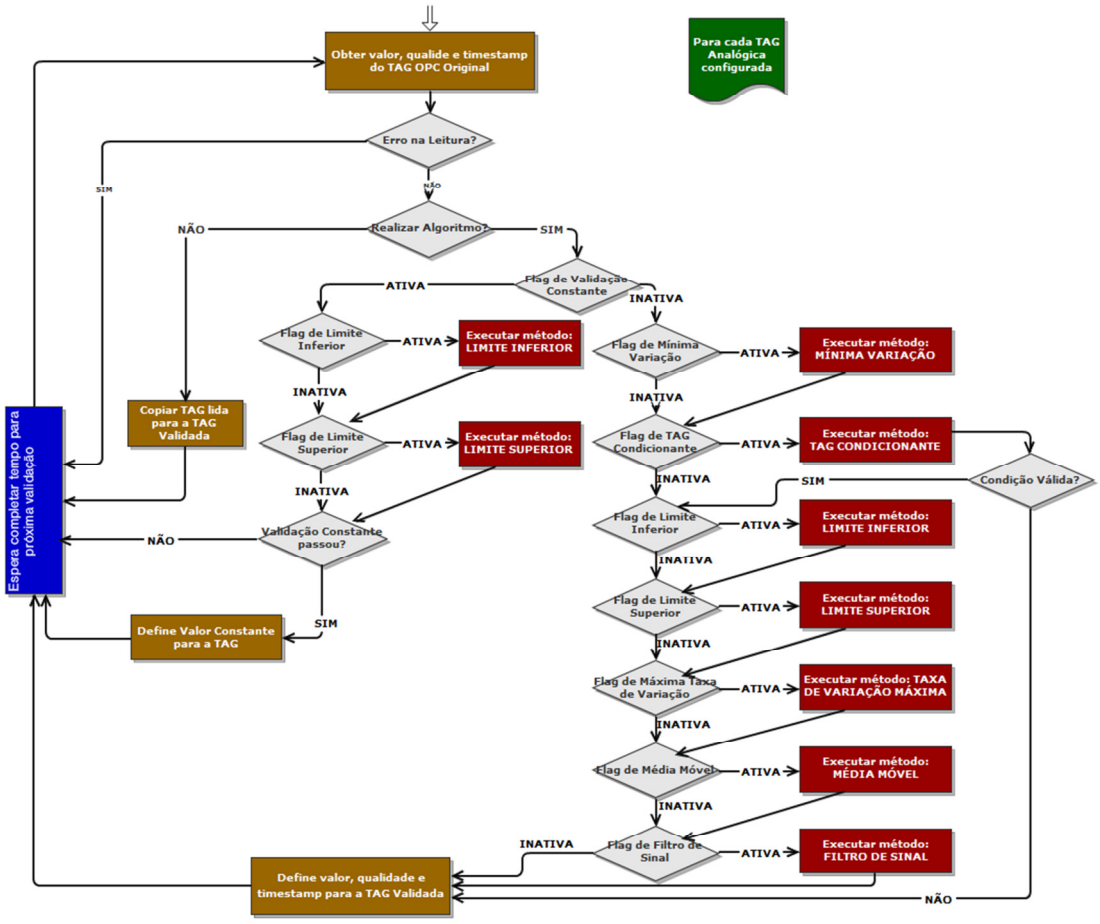


Figura 24 - Fluxo de TAG Analógica

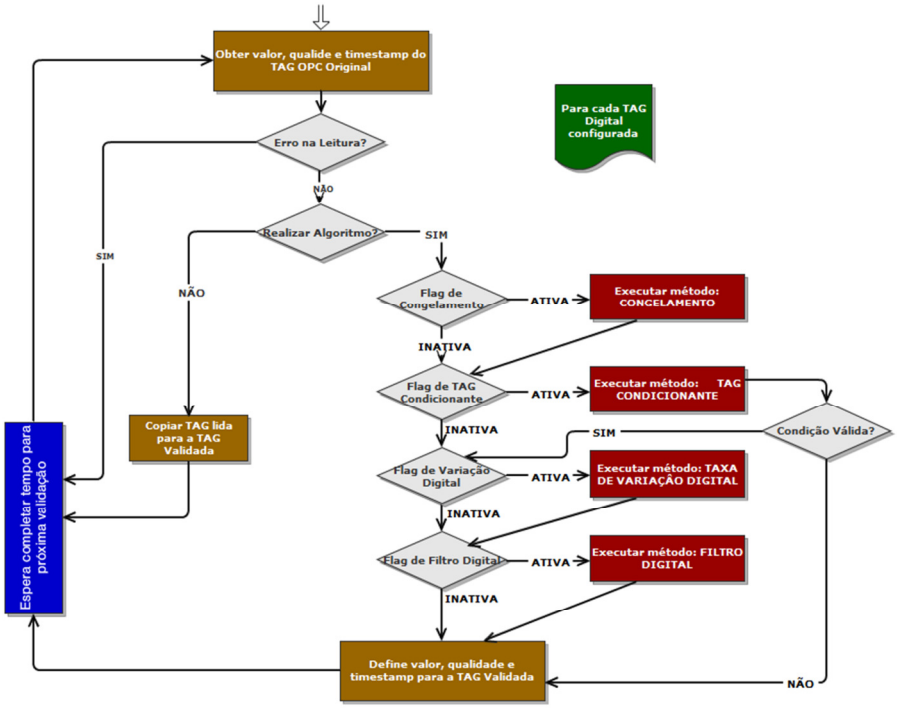


Figura 25 - Fluxo de TAG Digital

3.4.4: Operação e Aplicação: Execução

Feita toda configuração, estando a BDTR executando, basta conectarmos o servidor da ferramenta automatizadora de procedimentos (cliente OPC) à BDTR (servidor OPC).

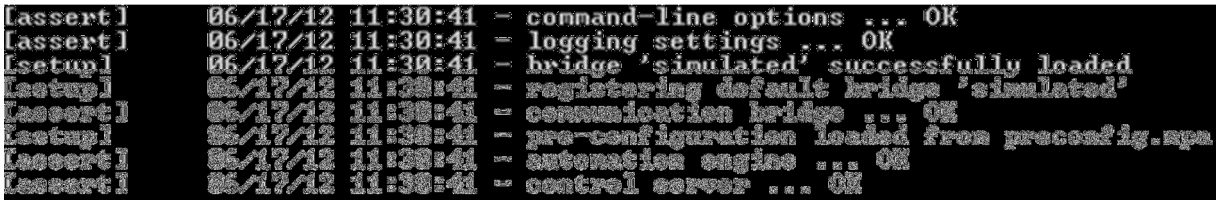


Figura 26 - Servidor da ferramenta executando

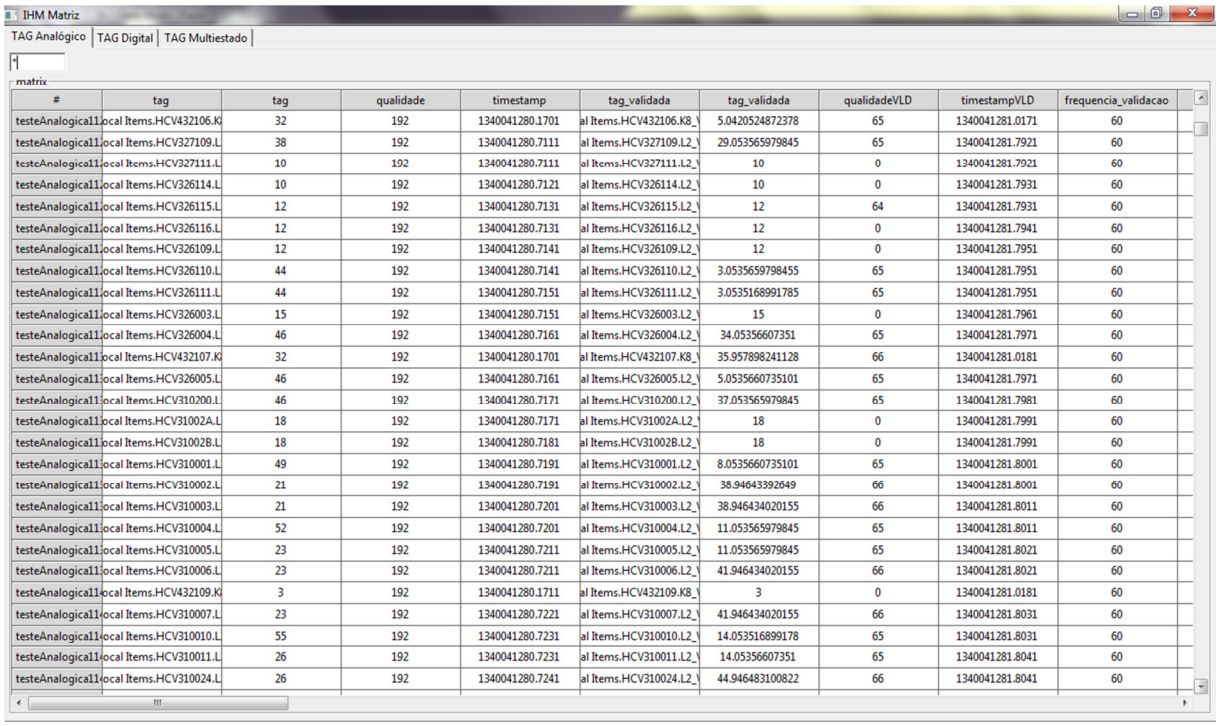
Assim, conectando o cliente da ferramenta ao servidor desta e exportando os fluxogramas para o servidor, estamos aptos para realizar a validação de sinais. Toda a configuração feita será avaliada e os valores serão escritos na BDTR para que, enfim, estejam disponíveis para outras aplicações.

Dessa forma, caso o fluxograma selecionado para executar seja o Fluxo de TAG Analógica. Todas as instâncias (com base nas linhas das tabelas da Configuração) serão criadas. Cada instância executará o *loop* do fluxo.

Primeiramente, são lidos os valores da TAG Origem do servidor OPC. Se não houver nenhum erro na leitura, o fluxo seguirá para a execução dos algoritmos. Supondo que o algoritmo de Mínima Variação esteja ativado na Configuração (“Flag de Mínima Validação” = *true*), o algoritmo definido na codificação será invocado e executado. Os outros algoritmos serão executados e, por fim, o valor validado será salvo. Aquela instância esperará sua próxima execução, definida pelo valor de frequência de validação, e reiniciará o *loop*.

3.4.5: Interface

Para permitir que valores da configuração sejam alterados durante a execução, uma interface simples foi criada. Esta realiza alterações nas instâncias e permite que o operador verifique o que está ocorrendo na planta. Assim, para realização dos testes, esta interface foi satisfatória.



#	tag	tag	qualidade	timestamp	tag_validada	tag_validada	qualidadeVLD	timestampVLD	frequencia_validacao
testeAnalogica11	ocal Items.HCV432106.K	32	192	1340041280.1701	al Items.HCV432106.K8_\	5.0420524872378	65	1340041281.0171	60
testeAnalogica11	ocal Items.HCV327109.L	38	192	1340041280.7111	al Items.HCV327109.L2_\	29.053565979845	65	1340041281.7921	60
testeAnalogica11	ocal Items.HCV327111.L	10	192	1340041280.7111	al Items.HCV327111.L2_\	10	0	1340041281.7921	60
testeAnalogica11	ocal Items.HCV326114.L	10	192	1340041280.7121	al Items.HCV326114.L2_\	10	0	1340041281.7931	60
testeAnalogica11	ocal Items.HCV326115.L	12	192	1340041280.7131	al Items.HCV326115.L2_\	12	64	1340041281.7931	60
testeAnalogica11	ocal Items.HCV326116.L	12	192	1340041280.7131	al Items.HCV326116.L2_\	12	0	1340041281.7941	60
testeAnalogica11	ocal Items.HCV326109.L	12	192	1340041280.7141	al Items.HCV326109.L2_\	12	0	1340041281.7951	60
testeAnalogica11	ocal Items.HCV326110.L	44	192	1340041280.7141	al Items.HCV326110.L2_\	3.0535659798455	65	1340041281.7951	60
testeAnalogica11	ocal Items.HCV326111.L	44	192	1340041280.7151	al Items.HCV326111.L2_\	3.0535168991785	65	1340041281.7951	60
testeAnalogica11	ocal Items.HCV326003.L	15	192	1340041280.7151	al Items.HCV326003.L2_\	15	0	1340041281.7961	60
testeAnalogica11	ocal Items.HCV326004.L	46	192	1340041280.7161	al Items.HCV326004.L2_\	34.05356607351	65	1340041281.7971	60
testeAnalogica11	ocal Items.HCV432107.K	32	192	1340041280.1701	al Items.HCV432107.K8_\	35.957898241128	66	1340041281.0181	60
testeAnalogica11	ocal Items.HCV326005.L	46	192	1340041280.7161	al Items.HCV326005.L2_\	5.0535660735101	65	1340041281.7971	60
testeAnalogica11	ocal Items.HCV310200.L	46	192	1340041280.7171	al Items.HCV310200.L2_\	37.053565979845	65	1340041281.7981	60
testeAnalogica11	ocal Items.HCV31002A.L	18	192	1340041280.7171	al Items.HCV31002A.L2_\	18	0	1340041281.7991	60
testeAnalogica11	ocal Items.HCV31002B.L	18	192	1340041280.7181	al Items.HCV31002B.L2_\	18	0	1340041281.7991	60
testeAnalogica11	ocal Items.HCV310001.L	49	192	1340041280.7191	al Items.HCV310001.L2_\	8.0535660735101	65	1340041281.8001	60
testeAnalogica11	ocal Items.HCV310002.L	21	192	1340041280.7191	al Items.HCV310002.L2_\	38.94643392649	66	1340041281.8001	60
testeAnalogica11	ocal Items.HCV310003.L	21	192	1340041280.7201	al Items.HCV310003.L2_\	38.946434020155	66	1340041281.8011	60
testeAnalogica11	ocal Items.HCV310004.L	52	192	1340041280.7201	al Items.HCV310004.L2_\	11.053565979845	65	1340041281.8011	60
testeAnalogica11	ocal Items.HCV310005.L	23	192	1340041280.7211	al Items.HCV310005.L2_\	11.053565979845	65	1340041281.8021	60
testeAnalogica11	ocal Items.HCV310006.L	23	192	1340041280.7211	al Items.HCV310006.L2_\	41.946434020155	66	1340041281.8021	60
testeAnalogica11	ocal Items.HCV432109.K	3	192	1340041280.1711	al Items.HCV432109.K8_\	3	0	1340041281.0181	60
testeAnalogica11	ocal Items.HCV310007.L	23	192	1340041280.7221	al Items.HCV310007.L2_\	41.946434020155	66	1340041281.8031	60
testeAnalogica11	ocal Items.HCV310010.L	55	192	1340041280.7231	al Items.HCV310010.L2_\	14.053516899178	65	1340041281.8031	60
testeAnalogica11	ocal Items.HCV310011.L	26	192	1340041280.7231	al Items.HCV310011.L2_\	14.05356607351	65	1340041281.8041	60
testeAnalogica11	ocal Items.HCV310024.L	26	192	1340041280.7241	al Items.HCV310024.L2_\	44.946483100822	66	1340041281.8041	60

Figura 27 – Interface IHM para a Monitoração e Validação de Sinais

Capítulo 4: Resultados

Os resultados apresentados neste capítulo referem-se à Monitoração e Validação de Sinais apresentados no capítulo anterior, desenvolvido na ferramenta automatizadora de procedimentos. Estes resultados foram utilizados para justificar que a ferramenta seria capaz de operar nas condições de campo. Após desenvolvidas as funcionalidades, diversos testes foram feitos para verificar desempenho e funcionamento correto, testados em diferentes cenários e condições.

Os cenários foram levantados pelo autor deste texto e por Diogo Pacheco (orientador da empresa). Os resultados apresentados foram obtidos no ambiente Radix e não no âmbito de operação. Estes testes permitiram avaliar a viabilidade da ferramenta proposta.

4.1: Resultados da Monitoração

Nesta seção, foi avaliado a eficiência e a funcionalidade do sistema em condições extremas, verificando sua robustez. Assim, situações acima da condição de operação foram utilizadas nos testes.

Além disso, é importante ressaltar que a ferramenta automatizadora de procedimentos pode operar de diversas maneiras, como por exemplo: leitura e escrita síncronas e assíncronas, como já citado anteriormente neste documento; leitura e escrita em blocos, fazendo uma chamada OPC para operações de diversas TAGs ou uma chamada por TAG; operar de maneira distribuída ou local.

Note que, como citado, estes testes ocorreram em ambiente corporativo, com máquinas *desktops* de uso cotidiano. Para a operação, foi sugerido a aquisição de uma máquina dedicada para execução da monitoração e validação de sinais e da BDTR, com desempenho muito melhor que as de uso comum.

4.1.1: Escopo do Teste

Para a realização dos testes, foram criadas diversas TAGs Analógicas, Digitais e Multiestados. Outras configurações foram necessárias também para que o

sistema operasse de maneira correta, como endereço para chamada de alarmes e registros em arquivos. Além disso, testes referentes a tanques (da monitoração total), que já estavam implementadas a esta altura foram acrescentados nos testes.

Assim, no total, estão sendo validadas cerca de 11000 TAGs digitais, 5000 TAGs analógicas e 750 TAGs multiestado. Esses valores foram feitos baseados em levantamentos de campo, nas refinarias, e a estrutura utilizada é bem semelhante a que existiria em campo. Além destes, foram feitos cálculos para 200 tanques, utilizando as TAGs criadas.

Um mecanismo de barreira foi implementado de maneira que garante que em um determinado ponto do fluxo todas as execuções lançadas em paralelo pela ferramenta estariam finalizadas. Também foram implementadas chamadas para abertura de arquivo de log e chamada de alerta (utilizando banco de dados), que foram executadas ou não, de acordo com o cenário de teste específico.

Neste teste, não verificou-se se a realização da validação estava sendo efetivamente correta pois busca-se exatamente a força bruta do sistema. A corretude da operação será verificada na seção de plano de testes (seção 4.2).

4.1.2: Cenários de Teste

Definiram-se alguns cenários para submeter aos testes. Tais cenários contemplam as formas de configuração do sistema. Com isso, espera-se identificar as condições que inviabilizam a monitoração, bem como aquelas que tendem a ser mais eficientes. Na tabela abaixo são listados os cenários submetidos aos testes:

Cenário	Instalalação	Configuração	Concorrência	LOGs	Alertas
1	Local	Com blocos	Sim	Não	Não
2	Local	Com blocos	Não	Não	Não
3	Local	Sem blocos	Sim	Não	Não
4	Local	Sem blocos	Não	Não	Não
5	Remoto	Com blocos	Sim	Não	Não
6	Remoto	Com blocos	Não	Não	Não

7	Remoto	Sem blocos	Sim	Não	Não
8	Remoto	Sem blocos	Não	Não	Não
9	Local	Com blocos	Sim	Sim	Não
10	Local	Com blocos	Sim	Sim	Sim
11	Remoto	Com blocos	Sim	Sim	Sim
Extra	Remoto	Com blocos	Sim	Sim	Não

Tabela 2 – Cenários de Teste da Monitoração

Com relação às formas de configuração, a instalação pode ser Local ou Remota: isso se refere a instalação do servidor OPC e a execução da ferramenta automatizadora de procedimentos. Se estão executando na mesma máquina, a execução é Local; se em máquinas distintas, Remoto.

Além disso, a ferramenta pode habilitar a comunicação em blocos ou não. A utilização de blocos em leituras e escritas pode melhorar, significantemente, o desempenho global da aplicação nos casos em que o tempo de resposta de uma leitura ou escrita de um único TAG é elevado. Há servidores que podem demandar mais de 2 segundos numa escrita enquanto outros o fazem quase que instantaneamente. Para a monitoração, esta configuração é sensível por haver um número muito grande de TAGs, sendo que cada décimo de segundo gasto em uma única TAG pode representar, globalmente, vários segundos.

Criaram-se ainda cenários para estressar o sistema a partir das inclusões de:

- Escritas de logs em um arquivo local à instalação da monitoração;
- Registro de alertas em uma base de dados;
- Acesso concorrente ao servidor OPC a partir de escritas contínuas de outro cliente, executado em uma terceira máquina (simulando a conexão com o SDCD).

A ferramenta utilizada é um sistema *monothread*, isto é, apenas uma operação do sistema ocorre em um dado instante de tempo, não havendo espaço para execução de diversos cenários concorrentemente. Os sistemas operacionais

atuais permitem a execução *multithread*, podendo várias execuções ocorrerem em paralelo.

Dessa forma, o cenário apresentado como “Extra” se refere a uma paralelização da operação da ferramenta, onde várias instâncias desta, cada uma com uma fração do sistema, executa de maneira paralela no Sistema Operacional. As frações se referem a execução da validação dos TAGs Analógicos, da dos TAGs Digitais e da dos TAGs Multiestados. Sendo assim, o tempo de execução deixa de ser a soma dos tempos individuais de cada função, mas o maior dos tempos de execução.

4.1.3: Resultados dos Testes

Para cada cenário definido foram executados 5 ciclos consecutivos. Este número foi definido empiricamente após observar-se poucas variações na duração dos ciclos após a inicialização (1º ciclo). Os testes foram repetidos pelo menos três vezes, verificando alterações mínimas entre as bateladas de teste. Por esta razão, os detalhes trazem informações de apenas um teste (5 ciclos) por cenário. Além disso, para contabilizar a média dos resultados, o primeiro ciclo foi ignorado, por não representar a operação contínua do sistema e sim seu *start-up*.

Na tabela abaixo são apresentados os tempos gastos para executar cada ciclo de todos os cenários testados. É possível notar uma pequena perda de desempenho quando não se utiliza operações em bloco. Por outro lado, o fato do servidor OPC ser concorrido por outro cliente não implicou em alterações de desempenho. Nos cenários em que se adicionou logs, cerca de 43 mil mensagens durante 5 ciclos foram registradas, com uma adição média de 2 segundos por ciclo. Nos cenários com banco de dados, aproximadamente 40 mil inserções num banco de dados com informações dos alertas foram realizadas ao longo de todos os ciclos.

Ciclo	**Cenários (tempos em segundos)**											
	1	2	3	4	5	6	7	8	9	10	11	Extra
1	25	29	35	33	27	27	135	136	29	36	76	31
2	14	14	18	16	15	14	33	35	17	22	33	11

3	14	14	19	18	15	14	38	36	16	18	31		11
4	14	13	23	16	15	15	42	33	16	18	34		12
5	13	16	20	17	15	15	35	33	17	24	32		11
Média	**14**	**14**	**20**	**17**	**15**	**15**	**37**	**34**	**17**	**21**	**33**		**11**

Tabela 3 – Resultados dos Cenários de Teste

O software de monitoração foi submetido a testes com carga suficiente para emular os cenários de produção da refinaria. Os testes abrangeram cenários explorando as formas de comunicação da monitoração e cenários que, lentamente, foram degradando o ambiente com outras funcionalidades concorrentes que não o cerne da monitoração.

O tempo médio de execução em todos os cenários práticos é inferior a 0,5 minuto. A monitoração não realizará verificações em intervalos inferiores a 1 minuto, como definido nos documentos de especificação. Com isso, mesmo se o sistema fosse degradado em 100%, este ainda conseguiria executar as funcionalidades dentro do intervalo limite estipulado. Ressalta-se ainda que o ambiente de execução de testes utilizou máquinas de uso diário e que os valores encontrados podem ser melhorados quando realizados no servidor específico para seu fim dentro do ambiente controlado da refinaria.

Pelos resultados apresentados, considera-se que a implementação do software de monitoração atendeu satisfatoriamente aos testes de desempenho aos quais foi submetida.

4.2: Plano de Testes

Como dito, os testes apresentados acima buscaram analisar a performance do sistema, aplicando a força bruta. Entretanto, não se garantia que o funcionamento adequado estavisse ocorrendo. Aliás, valores aleatórios eram geradas e não tinham valor real para o sistema. Deseja-se, porém, confirmar que o sistema funcione de maneira adequada.

Assim, um plano de testes foi criado, com diversos casos de teste que buscam confirmar e corroborar com as regras de negócio, evidenciando a corretude

do sistema. A idéia é que os casos de testes sirvam também como um guia para que o funcionamento adequado fosse verificado e o sistema desenvolvido homologado.

O resultado de um algoritmo depende muito da condição do sistema, com seus valores que são lidos, além da configuração em si, sendo difícil explicitar todas as possíveis situações em que o sistema poderia estar. Assim, apenas algumas foram criadas.

Por exemplo, um teste para o algoritmo de Mínima Variação é apresentado na tabela abaixo:

<table>
<tr><th colspan="2">Estado de Operação do Sistema</th><th rowspan="2">Configuração do Algoritmo</th><th rowspan="2">Reultado Esperado</th></tr>
<tr><th>TAG Original</th><th>TAG Validada</th></tr>
<tr><td>Valor = 375</td><td>Valor = 380</td><td>Frequência de Validação = 60</td><td rowspan="5">TAG Validada pelo Ciclo com Valor = 374 e qualidade = INCERTA. A estampa de tempo é feita no momento da escrita pelo servidor OPC.</td></tr>
<tr><td>Qualidade = BOA</td><td>Qualidade = BOA</td><td>Flag Mínima Variação = TRUE</td></tr>
<tr><td>Estampa de tempo = 10:15:00</td><td>Estampa de tempo = 10:00:00</td><td>Watchdog de Mín. Variação = 900</td></tr>
<tr><td colspan="2" rowspan="2"></td><td>Banda Morta de Mín. Variação = 6</td></tr>
<tr><td>Saída padrão = 3</td></tr>
</table>

Tabela 4 – Resultado de uma Validação

Além de casos de testes para os algoritmos, casos de testes para a inserção da configuração foram gerados, explicando o que cada entrada no sistema representava e seus tipos adequados.

Tudo isso, contribuiu para evidenciar que o sistema de Monitoração e Validação de Sinais efetivamente está de acordo com as especificações.

4.3: Discussão sobre os testes

A fase de teste é de fundamental importância para o sucesso de um projeto. Apesar de não existir softwares 100% livres de erros (em termos matemáticos), uma boa estratégia de teste pode garantir confiabilidade e segurança em níveis desejáveis aos sistemas.

Para a verificar a completude funcional e o atendimento aos requisitos, foi realizada logo após o desenvolvimento, em um ambiente interno da Radix, uma batelada de testes conhecidos como Testes Internos. Os resultados apresentados nas seções anteriores se referem a estes resultados.

Entretanto, durante o desenvolvimento, diversos testes foram realizados (nem sempre com a profundidade e a acurácia dos testes aqui apresentados) sem serem documentados, muitas vezes, apenas para testar funcionalidades locais, como de uma ou outra função. Nestes testes "pontuais", alguns problemas foram encontrados e foram solucionados conforme apareciam.

Alguns problemas, comum no desenvolvimento de softwares, decorreram de má implementação foram prontamente resolvidos. Outros, porém, levaram algum estudo e envolvimento de outros grupos e pessoas, inclusive dos desenvolvedores da ferramenta automatizadora de procedimentos.

O primeiro problema encontrado foi quanto ao tempo de ciclos. Nas mesmas condições dos testes apreentados, os ciclos levavam cerca de 180 segundos para completarem. Isto estava totalmente fora das especificações e, com certeza, não atenderia as condições de operação (que exige menos de 60 segundos por ciclo).

Revisando códigos, buscando soluções descobriu-se que o problema estava na primeira solução para emissão de alarmes, utilizando um banco de dados. A conexão era aberta e fechada a cada nova emissão de alarme o que causava um *overhead,* um processamento extra, que não deveria existir. A solução encontrada, foi abrir a conexão com o banco de dados e não fechá-la até que todos os ciclos estivessem terminado seu fluxo.

Isto foi descoberto no laboratório onde a ferramenta automatizadora de procedimentos foi desenvolvida, onde o autor deste texto e orientador estiveram para levantar possíveis problemas que estavam causando o alto processamento.

Outro problema que envolveu a equipe da ferramenta foi quanto à ponte OPC. Os dados passados pela BDTR são obtidos pelo cliente por subscrição. A ferramenta estava com problema para armezenar estes dados para serem utilizados posteriormente e não no momento de recebimento dos dados. Além disso, problemas com escrita em blocos foram encontrados.

Os problemas descritos no parágrafo passado levou algum tempo a mais para serem descobertos pois não dependiam do desenvolvimento da equipe Radix e sim de terceiros. A cada erro encontrado na ferramenta, uma nova versão de instalação era gerada e os testes eram refeitos para garantir o funcionamento do sistema. Conseguiu-se, por fim, resolver os problemas de maneira progressiva e, enfim, obter uma solução que atendia as especificações.

Apresentados os resultados dos testes, a ida de uma equipe Radix para a área de operações foi aprovada. O autor deste texto ficou dando suporte a equipe que se dirigiu à campo, na refinaria.

Entretanto, por problemas de segurança de TI na refinaria, a BDTR não conseguiu se comunicar com SDCDs e também com clientes OPC. Assim, grosso modo, apenas se repetiu os testes internos realizados na Radix com outro servidor OPC na refinaria para a comprovação da eficácia e corretude do sistema.

Capítulo 5: Conclusões e Perspectivas

Neste trabalho, discutiu-se sobre a implementação de um sistema de monitoração de dados que fornece dados validados para outros sistemas, a partir de uma série de regras. Além disso, apresentou-se um sistema de banco de dados em tempo real que centralizava toda a informação produzida pelo sistema de transferência e estocagem de uma refinaria de petróleo, que também mantém os dados validados.

Após a informação ser colhida e armazenada nesse sistema de banco de dados, algoritmos de validação atuam de maneira a “melhorar” o dado, evitando dados incongruentes e não corretos para o sistema. Assim, notadamente, o que se espera do sistema de validação não é apenas produzir dados mas produzir dados contribuam com a produção, fornecendo informações úteis para aumentar a eficiência do processo.

Neste sentido, é interessante citar Bill Gates: "A primeira regra de qualquer tecnologia utilizada nos negócios é que a automação aplicada a uma operação eficiente aumentará a eficiência. A segunda é que a automação aplicada a uma operação ineficiente aumentará a ineficiência”.

Assim, operadores sabem como melhorar e otimizar um processo. Entretanto, não podem fazê-lo de maneira constante e idêntica, sistematicamente. Como as regras são, de certa forma, empíricas, o aspecto humano atuando de maneira a otimizar o processo na planta é evidente.

Entretanto, o sistema desenvolvido é apenas uma parcela desse sistema de “refinamento de informações”, validando-se uma série de dados generalistas. Diversas outras funções ainda serão agregadas a solução, que ainda devem ser desenvolvidas, com cálculos mais intrínsecos ao negócio, como ligados a tanques.

É importante ressaltar que o sistema desenvolvido foi aplicado em refinaria, devido a uma necessidade específica. Porém, a aplicação dessas regras poderiam ser repassadas para diversos segmentos industriais, melhorando o desempenho e eficiência dos processos, como siderúrgicas, processos de manufatura e outros.

Bibliografia:

[1] Wikipedia, a enciclopédia livre, www.wikipedia.org (acessado em Junho de 2012)

[2] R. Sousa, História do Petróleo no Brasil, www.brasilescola.com (acessado em Junho de 2012)

[3] Radix – Engenharia e Software, www.radixeng.com.br (acessado em Junho de 2012)

[4] OPC Foundation, www.opcfoundation.org (acessado em Junho de 2012)

[5] OPC Data Access 3.00 Specification, OPC Foundation, Version 3.00, Março de 2003

[6] OPC Alarms & Events Specification, OPC Foundation, Version 1.02, Novembro de 1999

[7] Visual C# Developer Center, www.msdn.microsoft.com/vcsharp (acessado em Junho de 2012)

[8] R. Ierusalimschy, L. H. de Figueiredo, W. Celes, “Lua 5.0 Reference Manual”, PUC-Rio, 2003